EXHIBIT G

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

Exhibit B-10

Based on Headwater's apparent positions as to the scope of the patent's claims, as best they can be deciphered, the reference(s) charted below anticipate(s) or at least render(s) obvious the identified claims. The portions of the prior art system documents cited below are not exhaustive but are exemplary in nature. Additional citations may be found in the cover pleading.

This disclosure is not an admission that Samsung concedes any claim construction implied or suggested by Headwater's apparent positions as to the scope of the patent's claims, nor is it an admission by Samsung that any of its products are covered by or infringe the patent's claims, particularly when they are properly construed and applied. Samsung is not taking any claim construction positions through this disclosure, including whether the preamble is a limitation.

Samsung reserves the right to rely on additional citations or sources of evidence that also may be applicable, or that may become applicable in light of claim construction, changes in Headwater's infringement contentions, and/or information obtained during discovery as the case progresses. Samsung further reserves the right to amend or supplement this claim chart at a later date as more fully set forth in the Invalidity Contentions. For example, Defendants are currently in the process of taking discovery from non-parties including Nokia, HMD, Citrix, Google, Apple, and Microsoft. Accordingly, Defendants reserve the right to modify, amend, and/or supplement these contentions as information becomes available from non-parties.

Android is mobile device operating system that was initially released in September 2008. Applications (or "apps") can be installed on mobile devices that run Android. Any mobile device that predates the '184 patent, running an Android version with one or more apps that also predate the '184 patent, qualifies as prior art under at least pre-AIA 35 U.S.C. §§ 102(a)/(b). Such a device was known, used, offered for sale, and/or sold in the United States before the '184 patent.

Exemplary mobile devices that predate the '184 patent and were publicly available before the earliest possible priority date include: [1]

- HTC Dream/T-Mobile G1 (released September 2008)
- Samsung GT-I7500 Galaxy (released June 2009)

[1] *See, e.g.*, SAMSUNG_PRIORART0000001-334; SAMSUNG_PRIORART0005174-76; SAMSUNG_PRIORART0005177-317; SAMSUNG_PRIORART0005416-19; SAMSUNG_PRIORART0005420-23; SAMSUNG_PRIORART0005424-28; SAMSUNG_PRIORART0005429-44; SAMSUNG_PRIORART0005445-48; SAMSUNG_PRIORART0005449-52; SAMSUNG_PRIORART0005453-57; SAMSUNG_PRIORART0005458-71; SAMSUNG_PRIORART0005472-77; SAMSUNG_PRIORART0005478-84; SAMSUNG_PRIORART0005485-86; SAMSUNG_PRIORART0005488-5624.

- Nexus One (released January 2010)

Exemplary Android versions that predate the '184 patent and were publicly available before the earliest possible priority date include: [2]

- Android 1.0 (released September 2008)
- Android 1.1 (released February 2009)
- Android Cupcake (1.5) (released April 2009)
- Android Donut (1.6) (released September 2009)
- Android Eclair (2.0, 2.0.1, 2.1) (released October 2009 - January 2010)
- Android Froyo (2.2) (released May 20, 2010)

Android included files[3] such as ConnectivityManager, NetworkInfo, NetworkStateTracker, ThrottleManager, TrafficStats, ConnectivityManagerMobileTest, Socket, SocketTest, Power, PowerManager, PowerManagerTest, BatteryManager, and BatteryStats.

Exemplary apps that predate the '184 patent and were publicly available before the earliest possible priority date include: [4]

---

[2] See, e.g., SAMSUNG_PRIORART0003998; SAMSUNG_PRIORART0004085; SAMSUNG_PRIORART0004081; SAMSUNG_PRIORART0004086; SAMSUNG_PRIORART0004083; SAMSUNG_PRIORART0004084.
[3] The files listed are Java source files, so the filenames are, e.g., ConnectivityManager.java, NetworkInfo.java, etc., except as noted.
[4] See, e.g., SAMSUNG_PRIORART0000335-SAMSUNG_PRIORART0000383; POUZERATE0000001-POUZERATE0000261; GOOG-HEADWATER-000000001-123; HDWTR-GOOG00001-GOOG00013; SAMSUNG_PRIORART0005042; SAMSUNG_PRIORART0005062; SAMSUNG_PRIORART0005350; SAMSUNG_PRIORART0005351; SAMSUNG_PRIORART0005352; SAMSUNG_PRIORART0005353; SAMSUNG_PRIORART0005354; SAMSUNG_PRIORART0005355; SAMSUNG_PRIORART0005356; SAMSUNG_PRIORART0005357; SAMSUNG_PRIORART0005358; SAMSUNG_PRIORART0005359; SAMSUNG_PRIORART0005360; SAMSUNG_PRIORART0005361; SAMSUNG_PRIORART0005362; SAMSUNG_PRIORART0005363; SAMSUNG_PRIORART0005364; SAMSUNG_PRIORART0005046; SAMSUNG_PRIORART0005043; SAMSUNG_PRIORART0005044; SAMSUNG_PRIORART0005045; SAMSUNG_PRIORART0005054; SAMSUNG_PRIORART0005055; SAMSUNG_PRIORART0005056; SAMSUNG_PRIORART0005057; SAMSUNG_PRIORART0005058; SAMSUNG_PRIORART0005059; SAMSUNG_PRIORART0005060; SAMSUNG_PRIORART0005061; SAMSUNG_PRIORART0005318; SAMSUNG_PRIORART0005398;

- JuiceDefender (released January 2010) and its associated add-on application, UltimateJuice (collectively "JuiceDefender App")
- GreenPower (released March 2010)

As specific examples, an HTC Dream/T-Mobile G1, Samsung GT-I7500 Galaxy, or a Nexus One mobile device running any of Android versions 1.0- 2.2 by itself, or with the JuiceDefender or GreenPower applications installed, qualifies as prior art under at least pre-AIA 35 U.S.C. §§ 102(a)/(b). This device was known, used, offered for sale, and/or sold in the United States on or before May 20, 2010. At least the various documents cited in this claim chart describe the functionality of this device.

To the extent it is argued that Android Device with One or More Apps does not disclose or include each and every asserted claim limitation, either expressly or inherently, it would have been obvious to a POSITA to incorporate any of the teachings from the references identified in Exhibits B-01 through B-11, and B-B (whose exemplary citations for each limitation are incorporated herein) into Android Device with One or More Apps. Indeed, it would have been obvious to make such combinations and a POSITA would have had reason and motivation to make such combinations at least for reasons described herein and in the cover pleading.

'184 Claims	**Android Device with One or More Apps**
[1 pre] A wireless end-user device, comprising:	To the extent the preamble is a limitation, Android Device with One or More Apps discloses and/or renders obvious this element. For example, see the following passages and/or figures, as well as related disclosures: **Nexus One** The Nexus One is an example of an Android smartphone.

SAMSUNG_PRIORART0005047-53; SAMSUNG_PRIORART0005063-65; SAMSUNG_PRIORART0005066-74; SAMSUNG_PRIORART0005075-135; SAMSUNG_PRIORART0005136-52; SAMSUNG_PRIORART0005153-70; SAMSUNG_PRIORART0005171-73; SAMSUNG_PRIORART0005319-49; SAMSUNG_PRIORART0005365-74; SAMSUNG_PRIORART0005375-83; SAMSUNG_PRIORART0005384-94; SAMSUNG_PRIORART0005395-97; SAMSUNG_PRIORART0005487.

.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

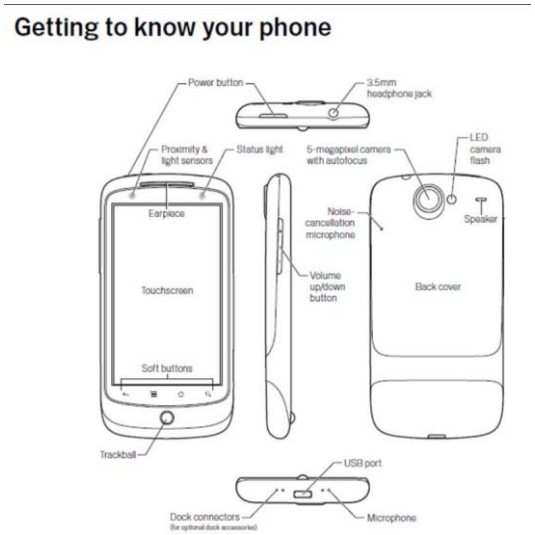
'184 Claims	Android Device with One or More Apps
	See, e.g., SAMSUNG_PRIORART0000001 (Nexus) at 17:  **HTC Dream / T-Mobile G1** The HTC Dream / T-Mobile G1 is an example of an Android smartphone. SAMSUNG_PRIORART0005184

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

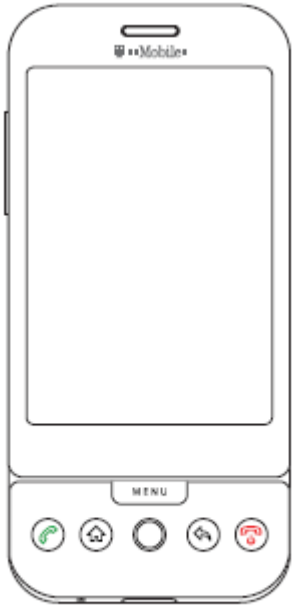
'184 Claims	Android Device with One or More Apps
	 **T-Mobile G1** SAMSUNG_PRIORART0005177 Android™ mobile technology platform R1.0 Document Rev 08 - September 8, 2008 Copyright 2008 © Google, Inc. All rights reserved. **Samsung GT-I7500 Galaxy**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

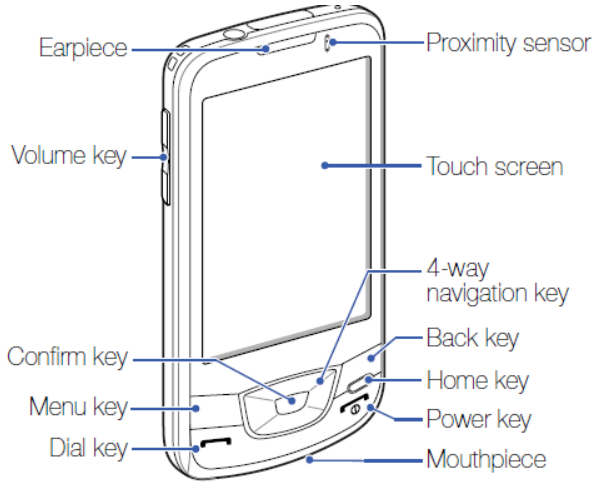
'184 Claims	Android Device with One or More Apps
	The Samsung GT-I7500 Galaxy is an example of an Android smartphone. SAMSUNG_PRIORART0005494 at PDF pgs. 2, 19:  **<u>JuiceDefender App</u>**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	JuiceDefender is a mobile application (or "app") intended to run on a mobile device, such an Android smartphone. The Nexus One is an example of an Android smartphone capable of running JuiceDefender. SAMSUNG_PRIORART0000379 (Latedroid) ("JuiceDefender saves battery power (lots of it!) by controlling the device data connection and/or WiFi … You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions.") SAMSUNG_PRIORART0000361 (Purdy) ("Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power.") **GreenPower App** GreenPower is a mobile application (or "app") intended to run on a mobile device, such an Android smartphone. The Nexus One is an example of an Android smartphone capable of running GreenPower. *See, e.g.,* POUZERATE0000015 (GDG Oslo) at 5:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	 See, e.g., POUZERATE0000002 (App Circus) at 9:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

<table>
<tr><th>'184 Claims</th><th>Android Device with One or More Apps</th></tr>
<tr><td>[1a] a wireless wide area network (WWAN) modem to communicate data for Internet service activities between the device and at least one WWAN, when configured for and connected to the at least one WWAN; and</td><td>Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures:

<u>Nexus One</u>

See, e.g., SAMSUNG_PRIORART0000001 (Nexus) at 332:
<table><tr><td>Cellular & wireless</td><td>Nexus One GSM phones compatible with 3G mobile networks from AT&T (U.S.) and Rogers Wireless (Canada):
3G UMTS bands I/II/V: 2100, 1900, 850 MHz

Nexus One GSM phones compatible with 3G mobile networks from T-Mobile (U.S.):
3G UMTS bands I/IV/VIII: 2100, 1700(AWS), 900 MHz

All Nexus One GSM phones:
HSDPA 7.2Mbps
HSUPA 2Mbps
GSM/EDGE 850, 900, 1800, 1900 MHz
Wi-Fi 802.11b/g
Bluetooth 2.1 + EDR
A2DP stereo Bluetooth</td></tr></table>
<u>HTC Dream / T-Mobile G1</u>

SAMSUNG_PRIORART0005202</td></tr>
</table>

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

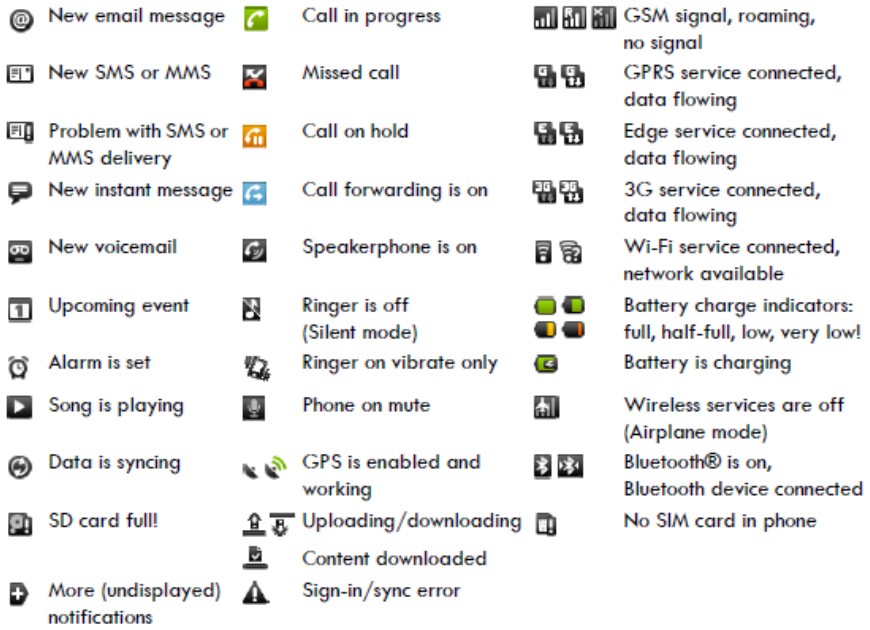
'184 Claims	Android Device with One or More Apps
	Mobile network settings • Select data roaming capability. • Select to connect only to 2G (slower) networks to save battery power. • Select a wireless operator network - Scan for all available networks, or select a network automatically. • Add or edit network Access Point Names (APNs) - Do not change this setting unless advised to do so by your wireless operator! SAMSUNG_PRIORART0005200

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

Notification and connection status icons

Along the top of your phone screen is the status bar. On the left side, icons will appear, notifying you of a new message, upcoming calendar event, alarm, or something else you should notice. On the right side of the status bar, you'll see connection status icons.

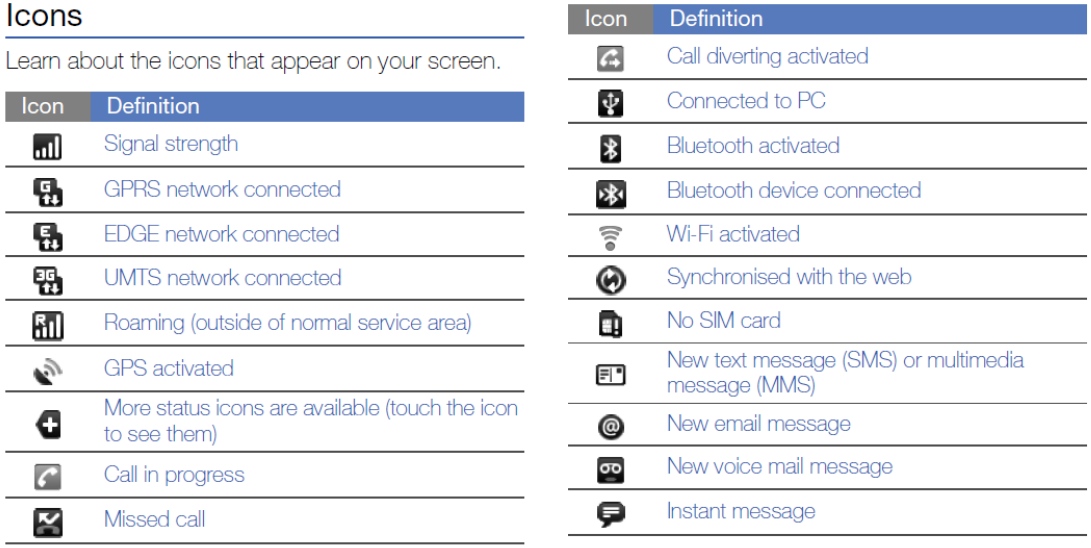


Samsung GT-I7500 Galaxy

SAMSUNG_PRIORART0005494 at PDF pg. 21:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps



Icons

Learn about the icons that appear on your screen.

Icon	Definition
	Signal strength
	GPRS network connected
	EDGE network connected
	UMTS network connected
	Roaming (outside of normal service area)
	GPS activated
	More status icons are available (touch the icon to see them)
	Call in progress
	Missed call

Icon	Definition
	Call diverting activated
	Connected to PC
	Bluetooth activated
	Bluetooth device connected
	Wi-Fi activated
	Synchronised with the web
	No SIM card
	New text message (SMS) or multimedia message (MMS)
	New email message
	New voice mail message
	Instant message

JuiceDefender App

SAMSUNG_PRIORART0000379 (Latedroid) ("JuiceDefender saves battery power (lots of it!) by controlling the device data connection and/or WiFi … You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions.").

SAMSUNG_PRIORART0000351 (Configuration-Translated) ("**APN:** activates / deactivates the APN connection, in its submenu we find the **MMS** button that activated configures the reception of MMS in the same way that we have the APN and **Prefer Wifi** that activated will try to connect first to this and if to five seconds does not find an available network will activate the APN. **WIFI:** activates / deactivates

'184 Claims	Android Device with One or More Apps
	the WIFI connection, in its submenu we find the following buttons, **Auto Disable** turns off the wifi in the case of not finding a network available to save battery, in case it is deactivated we will have to activate it manually; **Enable on Schedule / Peak / Screen will** activate the wifi as we have configured those buttons that we will see below.") SAMSUNG_PRIORART0000361 (Purdy) ("Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power."). SAMSUNG_PRIORART0000335 (Ruddock) ("Juice Defender is a battery conservation app. It uses various triggers, rules, and timers to control how often your device utilizes 3G/EDGE APN's (data connections) as well as WiFi. These data connections are the number one drainers of battery life when your phone is idle, so Juice Defender allows you to decide when, where, and how often you want them to be active."). **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Manage Mobile Network If this setting is selected, then Green Power will regularly turn on and off the Mobile Network connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Mobile Network as it is, never turning it on or off. Please note that in order for Green Power to turn on / off Mobile Network, this one has to be manually enabled by the user first in the phone settings (Wireless & networks → Mobile Network) or in Green Power settings (Global wireless settings → Mobile Network) . Green Power can't itself turn on Mobile Network as this is a limitation of the Android system for security and cost reasons."). **Android 1.0**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	GOOG-HEADWATER-00000040, SAMSUNG_PRIORART0005487, ConnectivityManager
	/**
	* Class that answers queries about the state of network connectivity. It also
	* notifies applications when network connectivity changes. Get an instance
	* of this class by calling
	* {@link android.content.Context#getSystemService(String)
	Context.getSystemService(Context.CONNECTIVITY_SERVICE)}.
	* <p>
	* The primary responsibilities of this class are to:
	*
	* - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
	* - Send broadcast intents when network connectivity changes
	* - Attempt to "fail over" to another network when connectivity to a network
	* is lost
	* - Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	/** {@hide} */
	public boolean setRadios(boolean turnOn) {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	try {
	return mService.setRadios(turnOn);
	} catch (RemoteException e) {
	return false;
	}
	}
	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {
	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {
	return false;
	}
	}
	
	SAMSUNG_PRIORART0005487, NetworkInfo
	/**
	* Indicates whether network connectivity exists or is in the process
	* of being established. This is good for applications that need to
	* do anything related to the network other than read or write data.
	* For the latter, call {@link #isConnected()} instead, which guarantees
	* that the network is fully usable.
	* @return {@code true} if network connectivity exists or is in the process
	* of being established, {@code false} otherwise.
	*/
	public boolean isConnectedOrConnecting() {
	return mState == State.CONNECTED || mState == State.CONNECTING;
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* Indicates whether network connectivity exists and it is possible to establish
	* connections and pass data.
	* @return {@code true} if network connectivity exists, {@code false} otherwise.
	*/
	public boolean isConnected() {
	return mState == State.CONNECTED;
	}
	/**
	* Indicates whether network connectivity is possible. A network is unavailable
	* when a persistent or semi-persistent condition prevents the possibility
	* of connecting to that network. Examples include
	*
	* - The device is out of the coverage area for any network of this type.
	* - The device is on a network other than the home network (i.e., roaming), and
	* data roaming has been disabled.
	* - The device's radio is turned off, e.g., because airplane mode is enabled.
	*
	* @return {@code true} if the network is available, {@code false} otherwise
	*/
	public boolean isAvailable() {
	return mIsAvailable;
	}
	
	/**
	* Sets if the network is available, ie, if the connectivity is possible.
	* @param isAvailable the new availability value.
	*
	* {@hide}
	*/

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	public void setIsAvailable(boolean isAvailable) {
	mIsAvailable = isAvailable;
	}
	
	public String getTypeName() {
	switch (mNetworkType) {
	case ConnectivityManager.TYPE_WIFI:
	return "WIFI";
	case ConnectivityManager.TYPE_MOBILE:
	return "MOBILE";
	default:
	return "<invalid>";
	}
	}
	
	SAMSUNG_PRIORART0005487, NetworkStateTracker
	/**
	* Turn the wireless radio off for a network.
	* @param turnOn {@code true} to turn the radio on, {@code false}
	*/
	public abstract boolean setRadio(boolean turnOn);
	
	/**
	* Returns an indication of whether this network is available for
	* connections. A value of {@code false} means that some quasi-permanent
	* condition prevents connectivity to this network.
	*/
	public abstract boolean isAvailable();

'184 Claims	Android Device with One or More Apps

	SAMSUNG_PRIORART0005487, ConnectivityService
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	// The WifiStateTracker should appear first in the list
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;
	
	mActiveNetwork = null;
	mNumDnsEntries = 0;
	mTestMode = SystemProperties.get("cm.test.mode").equals("true")
	&& SystemProperties.get("ro.build.type").equals("eng");
	for (NetworkStateTracker t : mNetTrackers)
	t.startMonitoring();
	// Constructing this starts it too
	mWifiWatchdogService = new WifiWatchdogService(context, mWifiStateTracker);
	}
	
	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing

'184 Claims	Android Device with One or More Apps
	* a connection to the preferred network is taken care of when we handle * the disconnect event from the non-preferred network * (see {@link #handleDisconnect(NetworkInfo)}). */ /** * Ensure that a network route exists to deliver traffic to the specified * host via the specified network interface. * @param networkType the type of the network over which traffic to the specified * host is to be routed * @param hostAddress the IP address of the host to which the route is desired * @return {@code true} on success, {@code false} on failure */ **Android 1.6** **SAMSUNG_PRIORART0005350, ConnectivityManager** /** * Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling * {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.) * - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* is lost
	* - Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	@SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
	public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED =
	"android.net.conn.BACKGROUND_DATA_SETTING_CHANGED";
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	public void setNetworkPreference(int preference) {
	try {
	mService.setNetworkPreference(preference);
	} catch (RemoteException e) {
	}
	}
	
	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {
	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	return false;
	}
	}
	
	/**
	* Returns the value of the setting for background data usage. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005350, NetworkInfo
	
	/**
	* Indicates whether network connectivity is possible:
	*/
	private boolean mIsAvailable;
	
	/**
	* Return a human-readable name describe the type of the network,
	* for example "WIFI" or "MOBILE".
	* @return the name of the network type
	*/
	public String getTypeName() {
	return mTypeName;
	}
	
	/**
	* Indicates whether network connectivity exists or is in the process
	* of being established. This is good for applications that need to
	* do anything related to the network other than read or write data.
	* For the latter, call {@link #isConnected()} instead, which guarantees

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* that the network is fully usable.
	* @return {@code true} if network connectivity exists or is in the process
	* of being established, {@code false} otherwise.
	*/
	public boolean isConnectedOrConnecting() {
	return mState == State.CONNECTED || mState == State.CONNECTING;
	}
	
	/**
	* Indicates whether network connectivity is possible. A network is unavailable
	* when a persistent or semi-persistent condition prevents the possibility
	* of connecting to that network. Examples include
	*
	* - The device is out of the coverage area for any network of this type.
	* - The device is on a network other than the home network (i.e., roaming), and
	* data roaming has been disabled.
	* - The device's radio is turned off, e.g., because airplane mode is enabled.
	*
	* @return {@code true} if the network is available, {@code false} otherwise
	*/
	public boolean isAvailable() {
	return mIsAvailable;
	}
	
	/**
	* Indicates whether the device is currently roaming on this network.
	* When {@code true}, it suggests that use of data on this network
	* may incur extra costs.
	* @return {@code true} if roaming is in effect, {@code false} otherwise.
	*/
	public boolean isRoaming() {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	return mIsRoaming;
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;
	
	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing
	* a connection to the preferred network is taken care of when we handle
	* the disconnect event from the non-preferred network
	* (see {@link #handleDisconnect(NetworkInfo)}).
	*/
	private void enforcePreference() {
	if (mActiveNetwork == null)

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	return;
	
	for (NetworkStateTracker t : mNetTrackers) {
	if (t == mActiveNetwork) {
	int netType = t.getNetworkInfo().getType();
	int otherNetType = ((netType == ConnectivityManager.TYPE_WIFI) ?
	ConnectivityManager.TYPE_MOBILE :
	ConnectivityManager.TYPE_WIFI);
	
	if (t.getNetworkInfo().getType() != mNetworkPreference) {
	NetworkStateTracker otherTracker = mNetTrackers[otherNetType];
	if (otherTracker.isAvailable()) {
	teardown(t);
	}
	}
	}
	}
	}
	
	/**
	* @see ConnectivityManager#getBackgroundDataSetting()
	*/
	public boolean getBackgroundDataSetting() {
	return Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, 1) == 1;
	}
	
	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}
	
	/**
	* See if the other network is available to fail over to.
	* If is not available, we enable it anyway, so that it
	* will be able to connect when it does become available,
	* but we report a total loss of connectivity rather than
	* report that we are attempting to fail over.
	*/
	NetworkInfo switchTo = null;
	if (newNet.isAvailable()) {
	mActiveNetwork = newNet;
	switchTo = newNet.getNetworkInfo();
	switchTo.setFailover(true);
	if (!switchTo.isConnectedOrConnecting()) {
	newNet.reconnect();
	}
	} else {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	newNet.reconnect();
	}
	
	if (info.getType() == ConnectivityManager.TYPE_MOBILE) {
	otherNet = mWifiStateTracker;
	} else /* info().getType() == TYPE_WIFI */ {
	otherNet = mMobileDataStateTracker;
	}
	
	int incrValue = ConnectivityManager.TYPE_MOBILE - ConnectivityManager.TYPE_WIFI;
	int stopValue = ConnectivityManager.TYPE_MOBILE + incrValue;
	
	SAMSUNG_PRIORART0005350, BatteryStats
	
	/**
	* A constant indicating a a wifi turn on timer
	*
	* {@hide}
	*/
	public static final int WIFI_TURNED_ON = 4;
	
	/**
	* A constant indicating a full wifi lock timer
	*
	* {@hide}
	*/
	public static final int FULL_WIFI_LOCK = 5;
	
	/**
	* A constant indicating a scan wifi lock timer

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* * {@hide} */ public static final int SCAN_WIFI_LOCK = 6; /** * A constant indicating a wifi multicast timer * * {@hide} */ public static final int WIFI_MULTICAST_ENABLED = 7; /** * A constant indicating an audio turn on timer * * {@hide} */ public static final int AUDIO_TURNED_ON = 7; /** * A constant indicating a video turn on timer * * {@hide} */ public static final int VIDEO_TURNED_ON = 8; /** * Include all of the data in the stats, including previously saved data. */ public static final int STATS_TOTAL = 0;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	/** * Include only the last run in the stats. */ public static final int STATS_LAST = 1; /** * Include only the current run in the stats. */ public static final int STATS_CURRENT = 2; /** * Include only the run since the last time the device was unplugged in the stats. */ public static final int STATS_UNPLUGGED = 3; public abstract void noteWifiTurnedOnLocked(); public abstract void noteWifiTurnedOffLocked(); public abstract void noteFullWifiLockAcquiredLocked(); public abstract void noteFullWifiLockReleasedLocked(); public abstract void noteScanWifiLockAcquiredLocked(); public abstract void noteScanWifiLockReleasedLocked(); public abstract void noteWifiMulticastEnabledLocked(); public abstract void noteWifiMulticastDisabledLocked(); /** * Returns the time in microseconds that the screen has been on while the device was * running on battery. * * {@hide} */

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	public abstract long getScreenOnTime(long batteryRealtime, int which); public static final int SCREEN_BRIGHTNESS_DARK = 0; public static final int SCREEN_BRIGHTNESS_DIM = 1; public static final int SCREEN_BRIGHTNESS_MEDIUM = 2; public static final int SCREEN_BRIGHTNESS_LIGHT = 3; public static final int SCREEN_BRIGHTNESS_BRIGHT = 4; public static final int DATA_CONNECTION_NONE = 0; public static final int DATA_CONNECTION_GPRS = 1; public static final int DATA_CONNECTION_EDGE = 2; public static final int DATA_CONNECTION_UMTS = 3; public static final int DATA_CONNECTION_OTHER = 4; /** * Returns the time in microseconds that wifi has been on while the device was * running on battery. * * {@hide} */ public abstract long getWifiOnTime(long batteryRealtime, int which); /** * Returns the time in microseconds that bluetooth has been on while the device was * running on battery. * * {@hide} */ public abstract long getBluetoothOnTime(long batteryRealtime, int which); /**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* Return whether we are currently running on battery. */ public abstract boolean getIsOnBattery(); /** * Returns the time that the radio was on for data transfers. * @return the uptime in microseconds while unplugged */ public abstract long getRadioDataUptime(); /** * Returns the current battery realtime in microseconds. * * @param curTime the amount of elapsed realtime in microseconds. */ public abstract long getBatteryRealtime(long curTime); /** * Returns the battery percentage level at the last time the device was unplugged from power, or * the last time it booted on battery power. */ public abstract int getDischargeStartLevel(); **Android 2.2** **GOOG-HEADWATER-00000029, SAMSUNG_PRIORART0005353, ConnectivityManager** /**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling * {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.) * - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network * is lost * - Provide an API that allows applications to query the coarse-grained or fine-grained * state of the available networks * */ * A change in network connectivity has occurred. A connection has either * been established or lost. The NetworkInfo for the affected network is * sent as an extra; it should be consulted to see what kind of * connectivity event occurred. /** * Broadcast Action: The setting for **background data usage** has changed * values. Use {@link #getBackgroundDataSetting()} to get the current value. * <p> * If an application uses the network in the background, it should listen * for this broadcast and stop using the background data if the value is * false. */

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	@SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION) public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED = "android.net.conn.BACKGROUND_DATA_SETTING_CHANGED"; /** * The Default Mobile data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_MOBILE = 0; /** * The Default WIFI data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_WIFI = 1; /** * Returns the value of the setting **for background data usage**. If false, * applications should not use the network if the application is not in the * foreground. Developers should respect this setting, and check the value * of this before performing any background data operations. * <p> * All applications that have background services that use the network * should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}. * * @return Whether background data usage is allowed. */

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	/**
	* Sets the persisted value for enabling/disabling Mobile data.
	*
	* @param enabled Whether the mobile data connection should be
	* used or not.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* @hide
	*/
	public void setMobileDataEnabled(boolean enabled) {
	try {
	mService.setMobileDataEnabled(enabled);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005353, NetworkStateTracker
	
	/**
	* Record the roaming status of the device, and if it is a change from the previous
	* status, send a notification to any listeners.
	* @param isRoaming {@code true} if the device is now roaming, {@code false}
	* if it is no longer roaming.
	*/
	protected void setRoamingStatus(boolean isRoaming) {
	if (isRoaming != mNetworkInfo.isRoaming()) {
	mNetworkInfo.setRoaming(isRoaming);
	Message msg = mTarget.obtainMessage(EVENT_ROAMING_CHANGED, mNetworkInfo);
	msg.sendToTarget();
	}
	}
	
	public static final int EVENT_ROAMING_CHANGED = 5;
	
	SAMSUNG_PRIORART0005353, ThrottleManager
	/**
	* Class that handles throttling. It provides read/write numbers per interface

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* and methods to apply throttled rates.
	* {@hide}
	*/
	
	/**
	* returns a long of the byte count either read or written on the named interface
	* for the period described. Direction is either DIRECTION_RX or DIRECTION_TX and
	* period may only be PERIOD_CYCLE for the current cycle (other periods may be supported
	* in the future). Ago indicates the number of periods in the past to lookup - 0 means
	* the current period, 1 is the last one, 2 was two periods ago..
	* {@hide}
	*/
	public long getByteCount(String iface, int direction, int period, int ago) {
	try {
	return mService.getByteCount(iface, direction, period, ago);
	} catch (RemoteException e) {
	return -1;
	}
	}
	
	/**
	* returns the number of bytes read+written after which a particular cliff
	* takes effect on the named iface. Currently only cliff #1 is supported (1 step)
	* {@hide}
	*/
	public long getCliffThreshold(String iface, int cliff) {
	try {
	return mService.getCliffThreshold(iface, cliff);
	} catch (RemoteException e) {
	return -1;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	}
	SAMSUNG_PRIORART0005353, ConnectivityManagerMobileTest
	
	// help function to verify 3G connection
	public void verifyCellularConnection() {
	NetworkInfo extraNetInfo = cmActivity.mNetworkInfo;
	assertEquals("network type is not MOBILE", ConnectivityManager.TYPE_MOBILE,
	extraNetInfo.getType());
	assertTrue("not connected to cellular network", extraNetInfo.isConnected());
	assertTrue("no data connection", cmActivity.mState.equals(State.CONNECTED));
	}
	
	// Wait for the Wifi state to be DISABLED
	waitForWifiState(WifiManager.WIFI_STATE_DISABLED, STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_WIFI, State.DISCONNECTED,
	STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_MOBILE, State.CONNECTED,
	STATE_TRANSITION_LONG_TIMEOUT);
	
	//Prepare for connectivity state verification
	NetworkInfo networkInfo = cmActivity.mCM.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
	cmActivity.setStateTransitionCriteria(ConnectivityManager.TYPE_MOBILE,
	networkInfo.getState(), NetworkState.DO_NOTHING,
	State.DISCONNECTED);
	networkInfo = cmActivity.mCM.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
	cmActivity.setStateTransitionCriteria(ConnectivityManager.TYPE_WIFI, networkInfo.getState(),
	NetworkState.TO_CONNECTION, State.CONNECTED);

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	// Wait for Wifi to be connected and mobile to be disconnected
	waitForNetworkState(ConnectivityManager.TYPE_WIFI, State.CONNECTED,
	STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_MOBILE, State.DISCONNECTED,
	STATE_TRANSITION_LONG_TIMEOUT);
	
	// Test case 5: test connectivity from 3G to airplane mode, then to 3G again
	
	// Test case 6: test connectivity with airplane mode Wifi connected
	
	SAMSUNG_PRIORART0005353, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	boolean noMobileData = !getMobileDataEnabled();
	for (int netType : mPriorityList) {
	switch (mNetAttributes[netType].mRadio) {
	case ConnectivityManager.TYPE_WIFI:
	if (DBG) Slog.v(TAG, "Starting Wifi Service.");
	WifiStateTracker wst = new WifiStateTracker(context, mHandler);
	WifiService wifiService = new WifiService(context, wst);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	wifiService.startWifi();
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = wst;
	wst.startMonitoring();

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	break;
	case ConnectivityManager.TYPE_MOBILE:
	mNetTrackers[netType] = new MobileDataStateTracker(context, mHandler,
	netType, mNetAttributes[netType].mName);
	mNetTrackers[netType].startMonitoring();
	if (noMobileData) {
	if (DBG) Slog.d(TAG, "tearing down Mobile networks due to setting");
	mNetTrackers[netType].teardown();
	}
	break;
	default:
	Slog.e(TAG, "Trying to create a DataStateTracker for an unknown radio type " +
	mNetAttributes[netType].mRadio);
	continue;
	}
	}
	
	/**
	* Sets the preferred network.
	* @param preference the new preference
	*/
	public synchronized void setNetworkPreference(int preference) {
	enforceChangePermission();
	if (ConnectivityManager.isNetworkTypeValid(preference) &&
	mNetAttributes[preference] != null &&
	mNetAttributes[preference].isDefault()) {
	if (mNetworkPreference != preference) {
	persistNetworkPreference(preference);
	mNetworkPreference = preference;
	enforcePreference();

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	}
	}
	
	/**
	* Return NetworkInfo for the active (i.e., connected) network interface.
	* It is assumed that at most one network is active at a time. If more
	* than one is active, it is indeterminate which will be returned.
	* @return the info for the active network, or {@code null} if none is
	* active
	*/
	public NetworkInfo getActiveNetworkInfo() {
	enforceAccessPermission();
	for (int type=0; type <= ConnectivityManager.MAX_NETWORK_TYPE; type++) {
	if (mNetAttributes[type] == null || !mNetAttributes[type].isDefault()) {
	continue;
	}
	NetworkStateTracker t = mNetTrackers[type];
	NetworkInfo info = t.getNetworkInfo();
	if (info.isConnected()) {
	if (DBG && type != mActiveDefaultNetwork) Slog.e(TAG,
	"connected default network is not " +
	"mActiveDefaultNetwork!");
	return info;
	}
	}
	return null;
	}
	
	// TODO - move this into the MobileDataStateTracker

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	int usedNetworkType = networkType;
	if(networkType == ConnectivityManager.TYPE_MOBILE) {
	if (!getMobileDataEnabled()) {
	if (DBG) Slog.d(TAG, "requested special network with data disabled - rejected");
	return Phone.APN_TYPE_NOT_AVAILABLE;
	}
	if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_MMS)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_MMS;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_SUPL)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_SUPL;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_DUN)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_DUN;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_HIPRI)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_HIPRI;
	}
	}
	
	int usedNetworkType = networkType;
	if (networkType == ConnectivityManager.TYPE_MOBILE) {
	if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_MMS)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_MMS;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_SUPL)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_SUPL;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_DUN)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_DUN;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_HIPRI)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_HIPRI;
	}
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* @see ConnectivityManager#getBackgroundDataSetting()
	*/
	public boolean getBackgroundDataSetting() {
	return Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, 1) == 1;
	}
	
	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {
	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA,
	allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}
	
	/**
	* @see ConnectivityManager#getMobileDataEnabled()
	*/

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	public boolean getMobileDataEnabled() {
	enforceAccessPermission();
	boolean retVal = Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.MOBILE_DATA, 1) == 1;
	if (DBG) Slog.d(TAG, "getMobileDataEnabled returning " + retVal);
	return retVal;
	}
	
	SAMSUNG_PRIORART0005353, BatteryStats
	
	/**
	* A class providing access to battery usage statistics, including information on
	* wakelocks, processes, packages, and services. All times are represented in microseconds
	* except where indicated otherwise.
	* @hide
	*/
	
	/**
	* A constant indicating a a wifi turn on timer
	*
	* {@hide}
	*/
	public static final int WIFI_TURNED_ON = 4;
	/**
	* A constant indicating an audio turn on timer
	*
	* {@hide}
	*/
	
	public static final int AUDIO_TURNED_ON = 7;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* A constant indicating a video turn on timer
	*
	* {@hide}
	*/
	public static final int VIDEO_TURNED_ON = 8;
	
	public static final int SIGNAL_STRENGTH_NONE_OR_UNKNOWN = 0;
	public static final int SIGNAL_STRENGTH_POOR = 1;
	public static final int SIGNAL_STRENGTH_MODERATE = 2;
	public static final int SIGNAL_STRENGTH_GOOD = 3;
	public static final int SIGNAL_STRENGTH_GREAT = 4;
	
	static final String[] SIGNAL_STRENGTH_NAMES = {
	"none", "poor", "moderate", "good", "great"
	};
	
	public static final int DATA_CONNECTION_NONE = 0;
	public static final int DATA_CONNECTION_GPRS = 1;
	public static final int DATA_CONNECTION_EDGE = 2;
	public static final int DATA_CONNECTION_UMTS = 3;
	public static final int DATA_CONNECTION_OTHER = 4;
	
	GOOG-HEADWATER-00000092, Google I/O 2009 - Coding for Life -- Battery Life, That Is (June 2, 2009)
	
	Google I/O, 2009

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

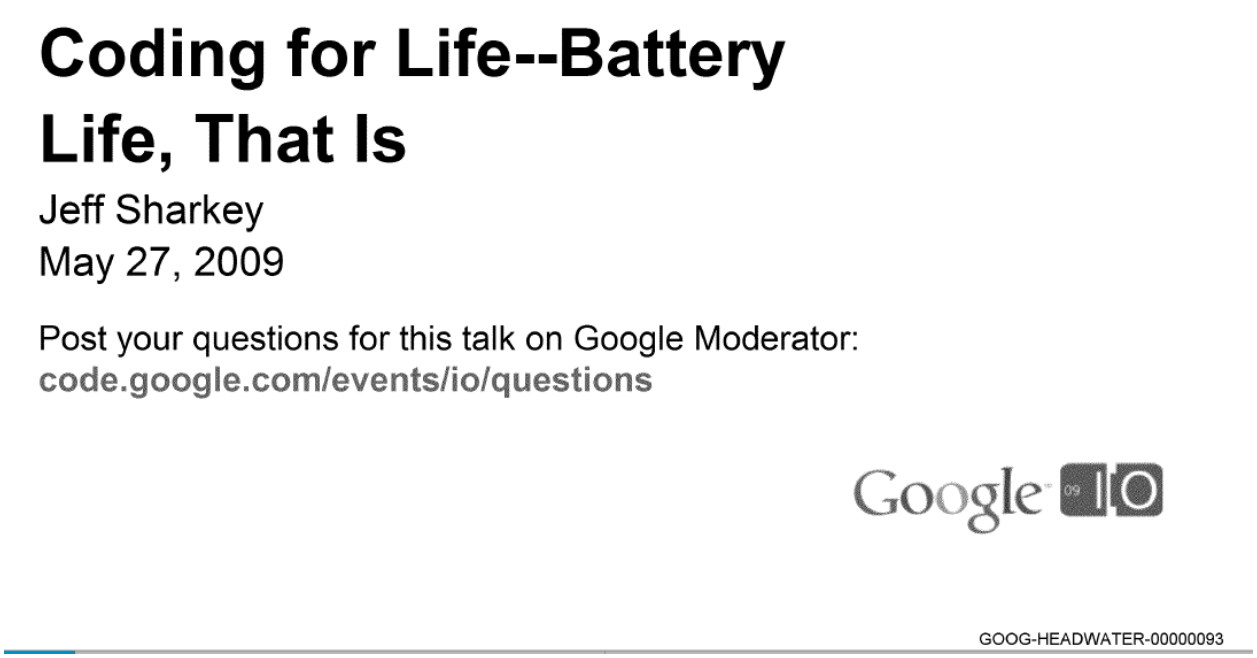
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 2: 

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 3: Why does this matter? ● Phones primarily run on battery power, and each device has a "**battery budget**" ○ When it's gone, it's gone ○ Apps need to work together to be good citizens of that shared resource ○ Current measured in mA, battery capacity in mAh ● HTC Dream: **1150mAh** ● HTC Magic: **1350mAh** ● Samsung I7500: **1500mAh** ● Asus Eee PC: **5800mAh** Google 09 I/O GOOG-HEADWATER-00000094

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

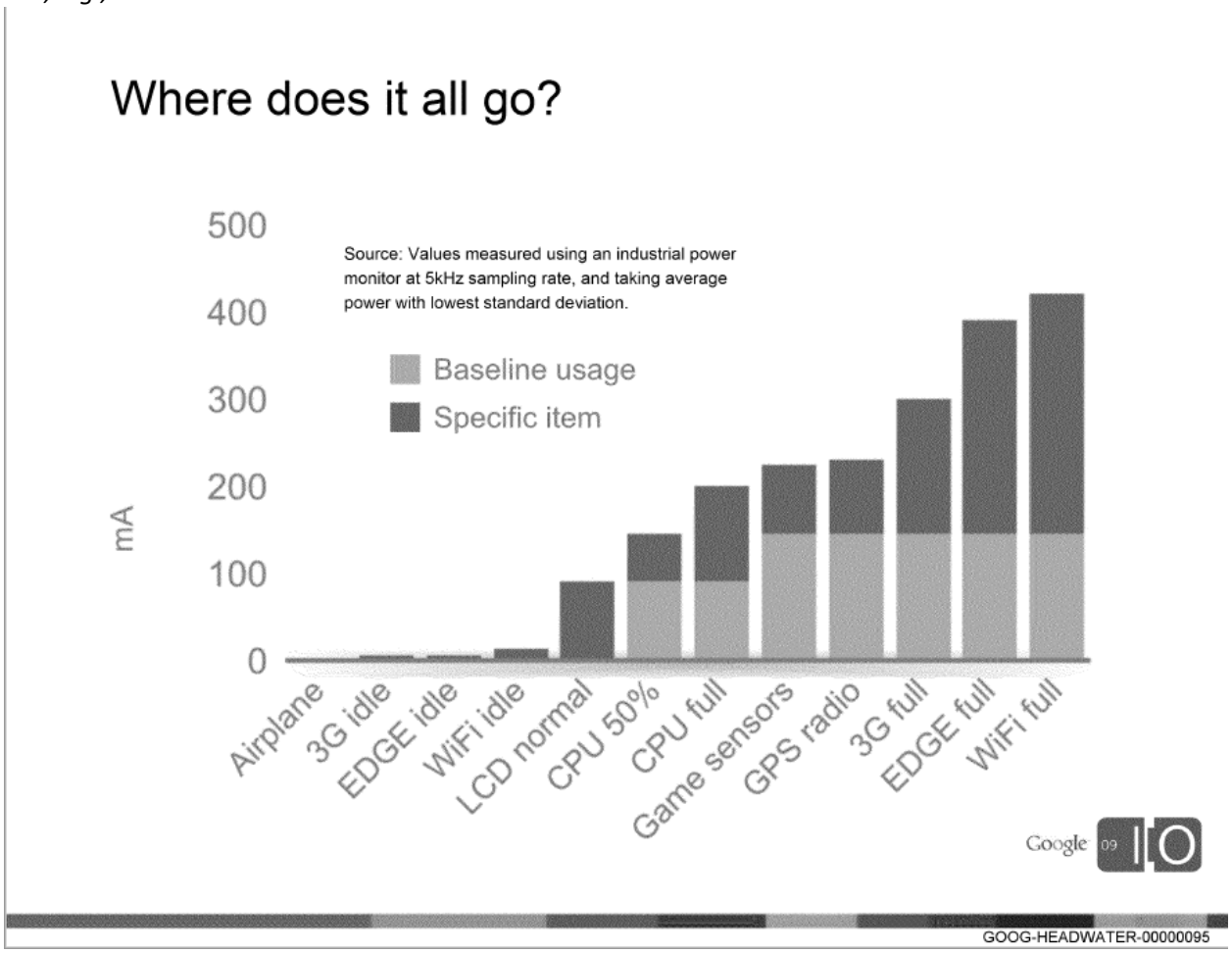
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 4: 

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

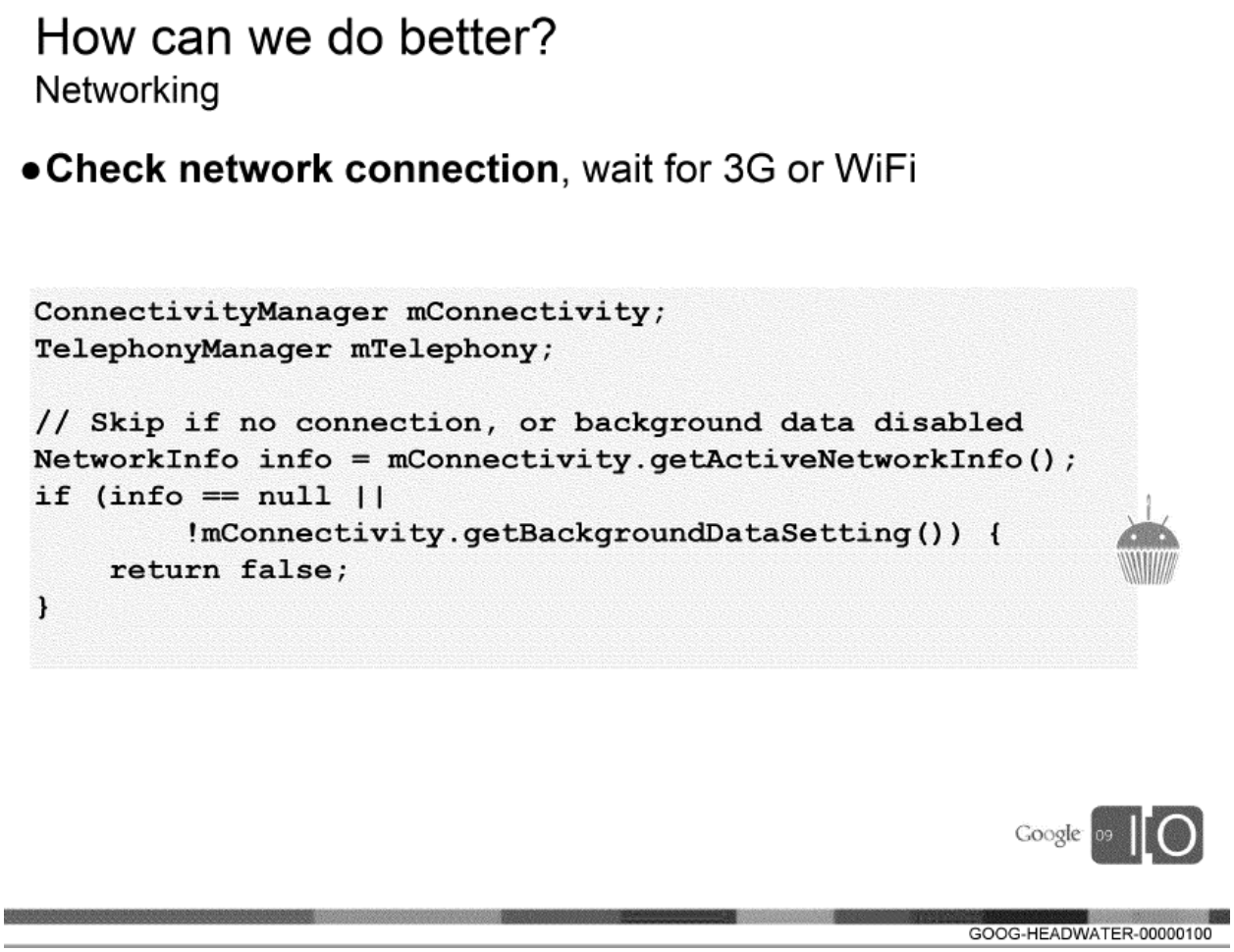
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 9:  *See, e.g.,* GOOG-HEADWATER-00000092 at 11:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	How can we do better?
	Networking
	● **Check network connection**, wait for 3G or WiFi
	`// Only update if WiFi or 3G is connected and not roaming`
	`int netType = info.getType();`
	`int netSubtype = info.getSubtype();`
	`if (netType == ConnectivityManager.TYPE_WIFI) {`
	`  return info.isConnected();`
	`} else if (netType == ConnectivityManager.TYPE_MOBILE`
	`  && netSubtype == TelephonyManager.NETWORK_TYPE_UMTS`
	`  && !mTelephony.isNetworkRoaming()) {`
	`  return info.isConnected();`
	`} else {`
	`  return false;`
	`}`
	Google 09 I/O
	GOOG-HEADWATER-00000101
	See, e.g., GOOG-HEADWATER-00000092 at 16:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

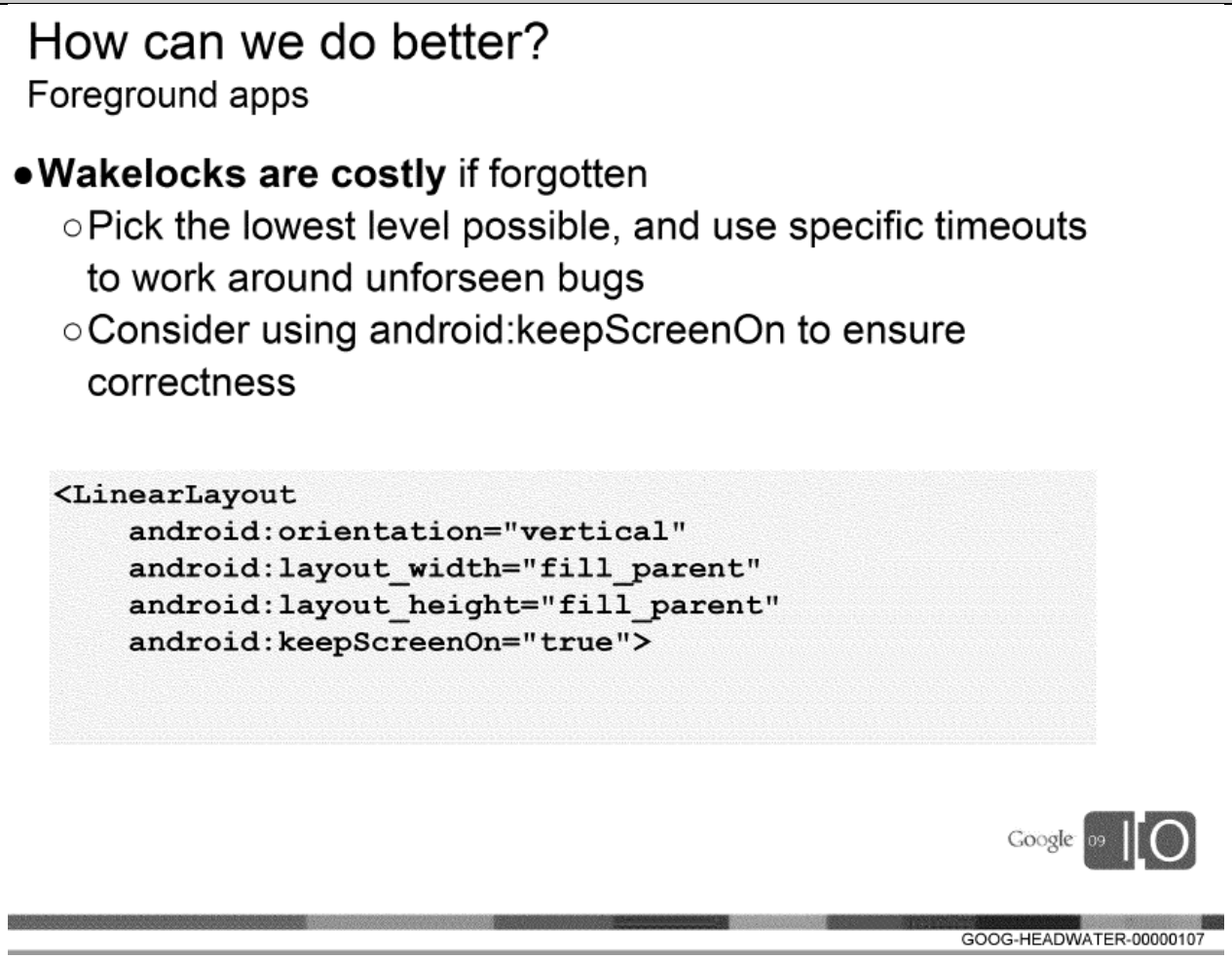
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 18:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	How can we do better? Foreground apps ● **Use coarse network location**, it's much cheaper ○ GPS: 25 seconds * 140mA = **1mAh** ○ Network: 2 seconds * 180mA = **0.1mAh** ● 1.5 uses AGPS when network available ● GPS time-to-fix varies wildly based on environment, and desired accuracy, and might outright fail ○ Just like wake-locks, location updates can continue after onPause(), so make sure to unregister ○ If all apps unregister correctly, user can leave GPS enabled in Settings Google 09 I/O GOOG-HEADWATER-00000109 *See, e.g.,* GOOG-HEADWATER-00000092 at 20:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

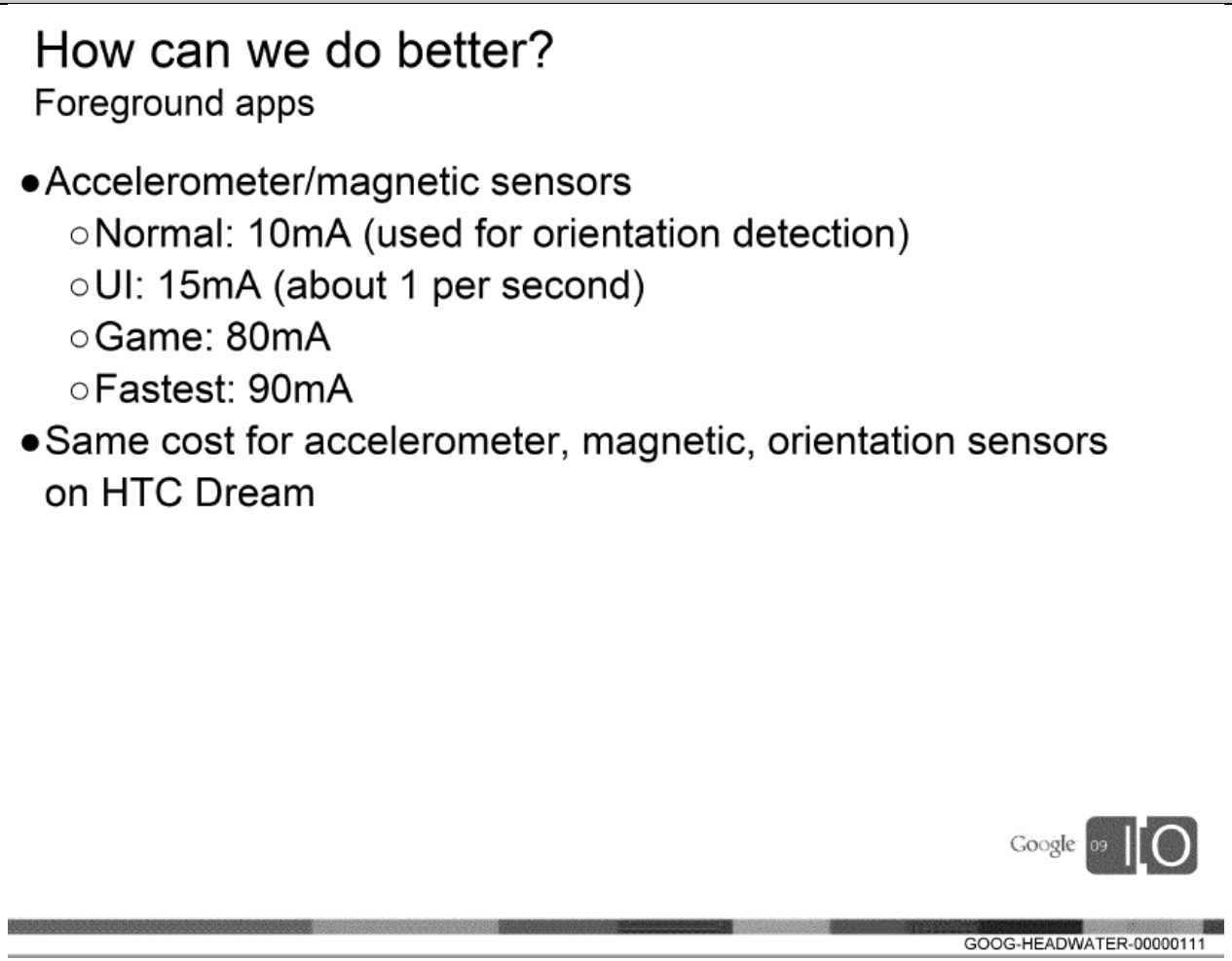
'184 Claims	Android Device with One or More Apps
	 *See, e.g.,* GOOG-HEADWATER-00000092 at 22:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	How can we do better? Background apps ● Services should be short-lived; these aren't daemons ○ Each process costs 2MB and risks being killed/restarted as foreground apps need memory ○ Otherwise, keep memory usage low so you're not the first target ● Trigger wake-up through AlarmManager or with <receiver> manifest elements ○ stopSelf() when finished Google I/O 09 GOOG-HEADWATER-00000113 *See, e.g.,* GOOG-HEADWATER-00000092 at 26:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

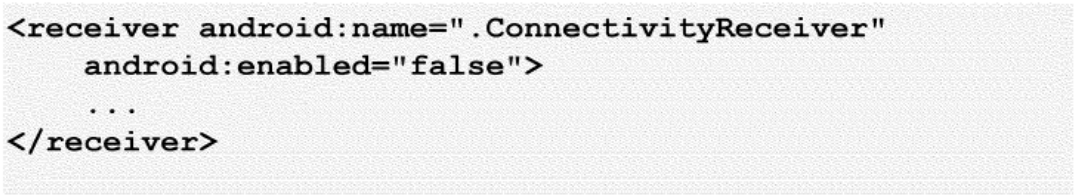
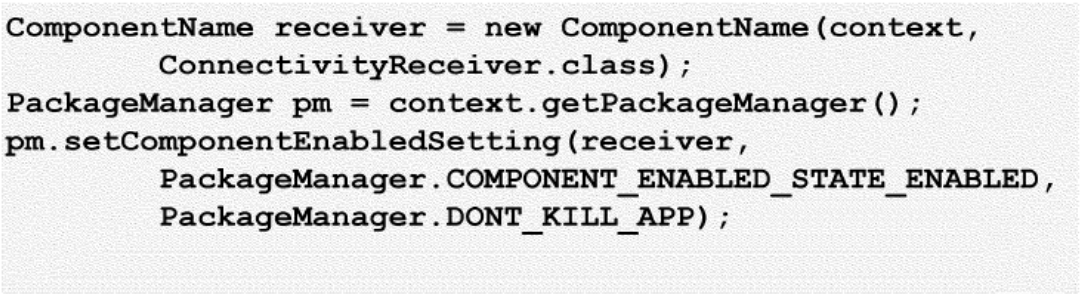
'184 Claims	Android Device with One or More Apps
	How can we do better? Background apps ● Dynamically enabling/disabling <receiver> components in manifest, especially when no-ops   Google 09 I/O GOOG-HEADWATER-00000117

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	See, e.g., GOOG-HEADWATER-00000092 at 27:
	How can we do better?
	Background apps
	● Checking current battery and network state before running a full update
	`public void onCreate() {`
	`  // Register for sticky broadcast and send default`
	`  registerReceiver(mReceiver, mFilter);`
	`  mHandler.sendEmptyMessageDelayed(MSG_BATT, 1000);`
	`}`
	`IntentFilter mFilter =`
	`  new IntentFilter(Intent.ACTION_BATTERY_CHANGED);`
	`BroadcastReceiver mReceiver = new BroadcastReceiver() {`
	`  public void onReceive(Context context, Intent intent) {`
	`  // Found sticky broadcast, so trigger update`
	`  unregisterReceiver(mReceiver);`
	`  mHandler.removeMessages(MSG_BATT);`
	`  mHandler.obtainMessage(MSG_BATT, intent).sendToTarget();`
	`  }`
	`};`
	Google 09 IO
	GOOG-HEADWATER-00000118
	See, e.g., GOOG-HEADWATER-00000092 at 29:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

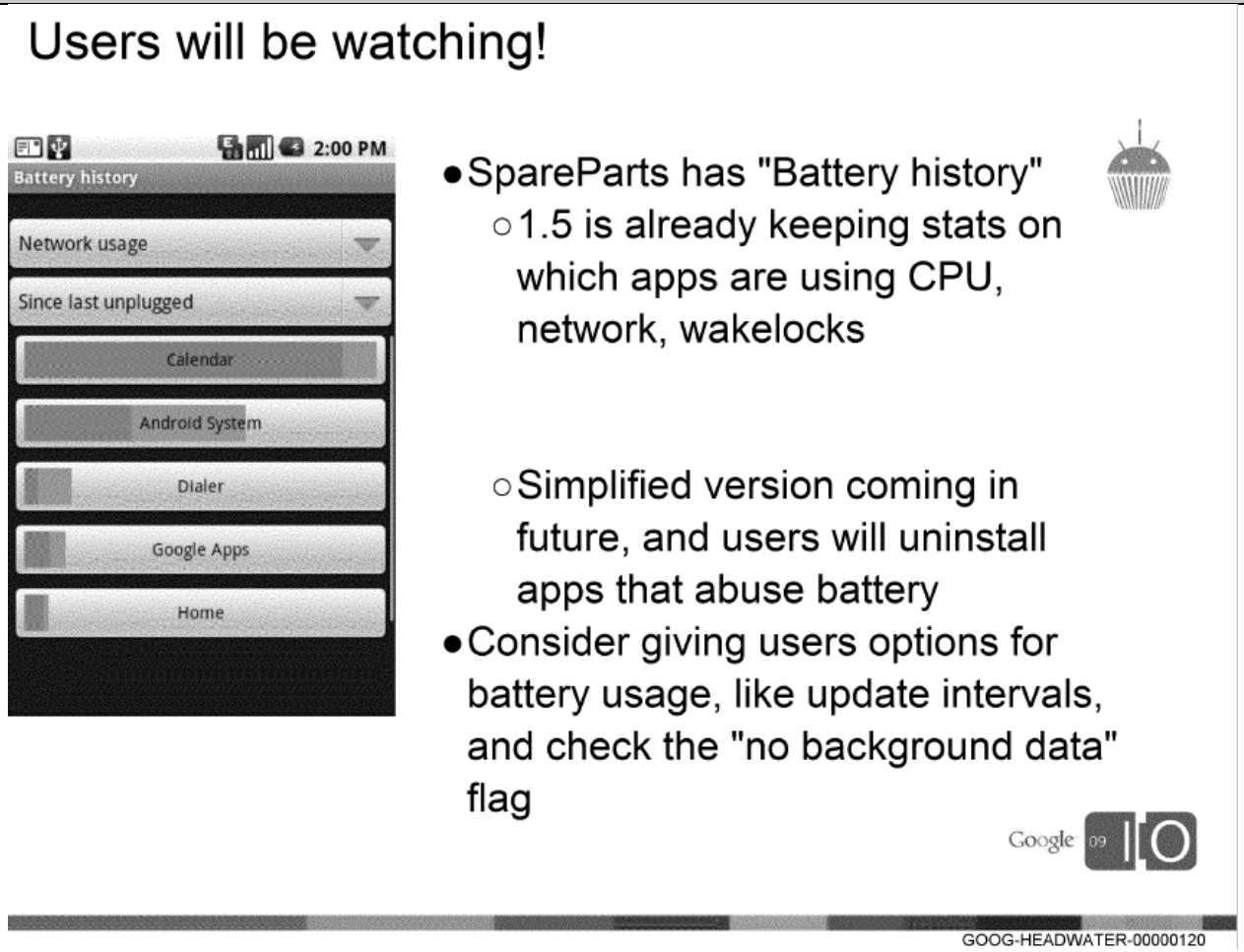
'184 Claims	Android Device with One or More Apps
	 *See, e.g.,* GOOG-HEADWATER-00000092 at 30:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

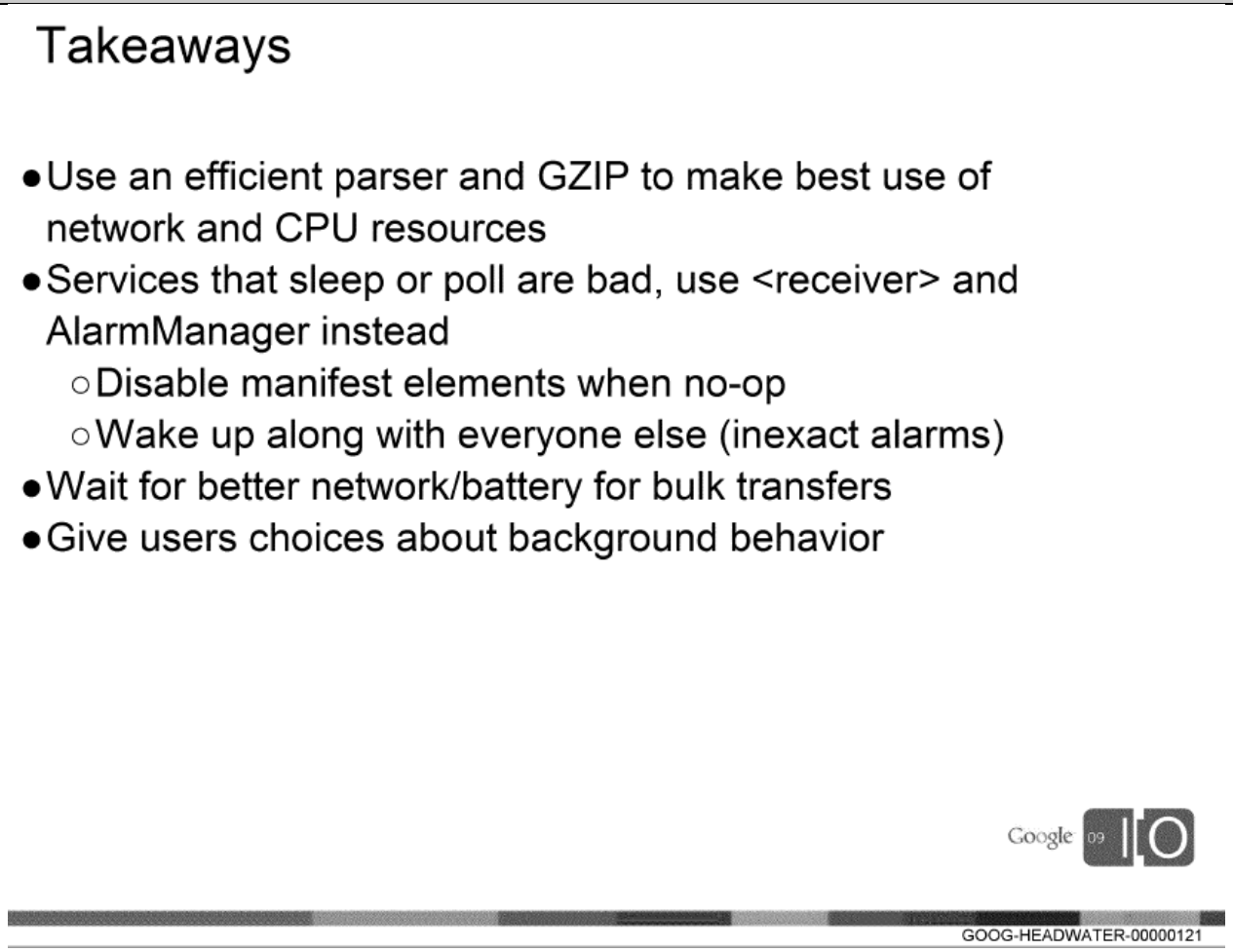
'184 Claims	Android Device with One or More Apps
	
[1b] one or more processors configured	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
to, for a time when data communication for Internet service activities is provided by the WWAN modem,	*See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 331: Processor | Qualcomm QSD 8250, 1 GHz *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 320: **Accounts & sync settings screen** **Background data** Check to permit applications to synchronize data in the background, whether or not you are actively working in them. Unchecking this setting can save battery power and lowers (but does not eliminate) data use. **Auto-sync** Check to permit applications to synchronize data on their own schedule. If you uncheck this setting, you must touch an account in the list on this screen, press **Menu**, and touch **Sync now** to synchronize data for that account. Synchronizing data automatically is disabled if **Background data** is unchecked. In that case, the **Auto-sync** checkbox is dimmed. SAMSUNG_PRIORART0000001 (Nexus) at 115-116 ("You can configure background data use and synchronization options for all of the applications on your phone. You can also configure what kinds of data you synchronize for each account. Some applications, such as Gmail and Calendar, have their own synchronization settings."). SAMSUNG_PRIORART0000001 (Nexus) at 115-116:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

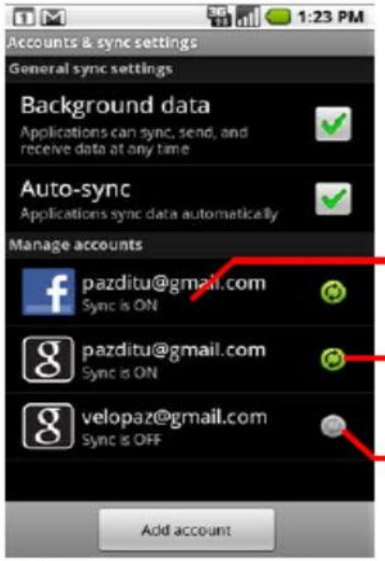
'184 Claims	Android Device with One or More Apps
	The screen displays your current sync settings and a list of your current accounts.  indicates that some or all of an account's information is configured to sync automatically with your phone. indicates that none of an account's information is configured to sync automatically with your phone. *See also, e.g., [1a].*
[1c] classify whether a particular application associated with an Internet service access request, and capable of both interacting with a	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 320:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

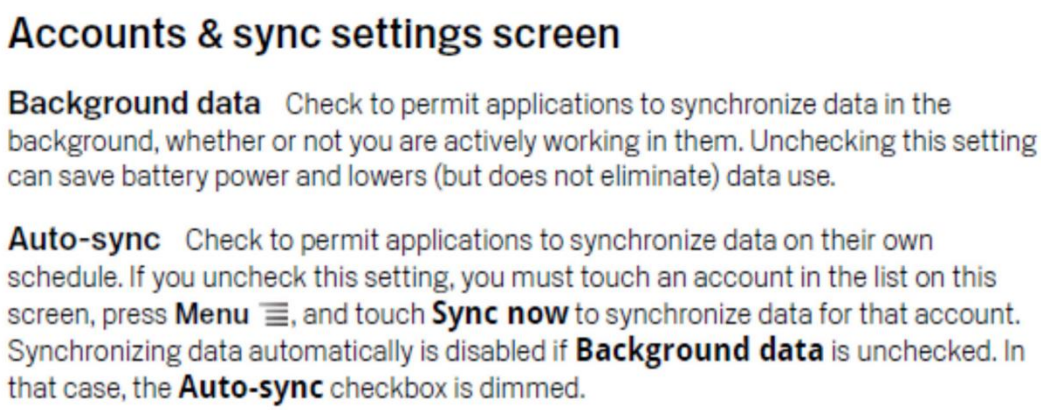
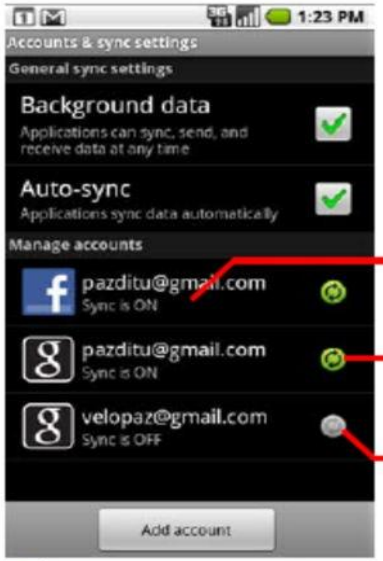
'184 Claims	Android Device with One or More Apps
user in a user interface foreground of the device, and at least some Internet service activities when not interacting with a user in the device user interface foreground, is interacting with the user in the device user interface foreground,	 SAMSUNG_PRIORART0000001 *(*Nexus) at 115-116 ("You can configure background data use and synchronization options for all of the applications on your phone. You can also configure what kinds of data you synchronize for each account. Some applications, such as Gmail and Calendar, have their own synchronization settings."). SAMSUNG_PRIORART0000001 (Nexus) at 115-116:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

The screen displays your current sync settings and a list of your current accounts.



indicates that some or all of an account's information is configured to sync automatically with your phone.

indicates that none of an account's information is configured to sync automatically with your phone.

Android 1.0

SAMSUNG_PRIORART0005487, Activity

* <p>Activities in the system are managed as an activity stack.
* When a new activity is started, it is placed on the top of the stack

'184 Claims	Android Device with One or More Apps
	* and becomes the running activity -- the previous activity always remains * below it in the stack, and will not come to the foreground again until * the new activity exits. * * An activity has essentially four states: * * - If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * - If an activity has lost focus but is still visible (that is, a new non-full-sized * or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme * low memory situations. * - If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden * and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be * completely restarted and restored to its previous state. * * * The following diagram shows the important state paths of an Activity. * The square rectangles represent callback methods you can implement to * perform operations when the Activity moves between states. The colored * ovals are major states the Activity can be in.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* * <p><img src="../../../images/activity_lifecycle.png" * The entire lifetime of an activity happens between the first call * to {@link android.app.Activity#onCreate} through to a single final call * to {@link android.app.Activity#onDestroy}. An activity will do all setup * of "global" state in onCreate(), and release all remaining resources in * onDestroy(). For example, if it has a thread running in the background * to download data from the network, it may create that thread in onCreate() * and then stop the thread in onDestroy(). * * The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user. * * The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* intent is delivered -- so the code in these methods should be fairly * lightweight. * * * The entire lifecycle of an activity is defined by the following * Activity methods. All of these are hooks that you can override * to do appropriate work when the activity changes state. All * activities will implement {@link android.app.Activity#onCreate} * to do their initial setup; many will also implement * {@link android.app.Activity#onPause} to commit changes to data and * otherwise prepare to stop interacting with the user. You should always * call up to your superclass when implementing these methods. * <tr><th colspan="2" align="left" border="0">{@link android.app.Activity#onStop onStop()}</th> * <td>Called when the activity is no longer visible to the user, because * another activity has been resumed and is covering this one. This * may happen either because a new activity is being started, an existing * one is being brought in front of this one, or this one is being * destroyed. * Followed by either <code>onRestart()</code> if * this activity is coming back to interact with the user, or * <code>onDestroy()</code> if this activity is going away.</td> * <td align="center">Yes</td> * <td align="center"><code>onRestart()</code> or * <code>onDestroy()</code></td> * </tr> * The Android system attempts to keep application process around for as

'184 Claims	Android Device with One or More Apps
	* long as possible, but eventually will need to remove old processes when * memory runs low. As described in Activity * Lifecycle, the decision about which process to remove is intimately * tied to the state of the user's interaction with it. In general, there * are four states a process can be in based on the activities running in it, * listed here in order of importance. The system will kill less important * processes (the last ones) before it resorts to killing more important * processes (the first ones). * * * <p>The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has * reached a memory paging state, so this is required in order to keep the user * interface responsive. * <p>A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * <p>A background activity (an activity that is not visible to * the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its * {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* <p>An empty process is one hosting no activities or other * application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * /** * Called as part of the activity lifecycle when an activity is going into * the background, but has not (yet) been killed. The counterpart to * {@link #onResume}. * * <p>When activity B is launched in front of activity A, this callback will * be invoked on A. B will not be created until A's {@link #onPause} returns, * so be sure to not do anything lengthy here. * * <p>This callback is mostly used for saving any persistent state the * activity is editing, to present a "edit in place" model to the user and * making sure nothing is lost if there are not enough resources to start * the new activity without first killing this one. This is also a good * place to do things like stop animations and other things that consume a * noticeable mount of CPU in order to make the switch to the next activity * as fast as possible, or to close resources that are exclusive access * such as the camera. * * <p>In situations where the system needs more memory it may kill paused

'184 Claims	Android Device with One or More Apps
	* processes to reclaim resources. Because of this, you should be sure * that all of your state is saved by the time you return from * this function. In general {@link #onSaveInstanceState} is used to save * per-instance state in the activity and this method is used to store * global persistent data (in content providers, files, etc.) * * After receiving this call you will usually receive a following call * to {@link #onStop} (after the next activity has been resumed and * displayed), however in some cases there will be a direct call back to * {@link #onResume} without going through the stopped state. * * Derived classes must call through to the super class's * implementation of this method. If they do not, an exception will be * thrown. * * @see #onResume * @see #onSaveInstanceState * @see #onStop */ **SAMSUNG_PRIORART0005487, ActivityManager** /** * Return a list of the tasks that are currently running, with * the most recent being first and older ones after in order. Note that * "running" does not mean any of the task's code is currently loaded or * activity -- the task may have been frozen by the system, so that it * can be restarted in its previous state when next brought to the * foreground. *

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* @param maxNum The maximum number of entries to return in the list. The
	* actual number returned may be smaller, depending on how many tasks the
	* user has started.
	*
	* @return Returns a list of RunningTaskInfo records describing each of
	* the running tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RunningTaskInfo> getRunningTasks(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningTaskInfo>)ActivityManagerNative.getDefault()
	.getTasks(maxNum, 0, null);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Set to true if the service has asked to run as a foreground process.
	*/
	public boolean foreground;
	
	Android 1.6
	
	SAMSUNG_PRIORART0005350, Fundamentals.jd

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Activities An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others. Each one is implemented as a subclass of the {@link android.app.Activity} base class. An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one. Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen. The visual content of the window is provided by a hierarchy of views — objects derived from the base {@link android.view.View} class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space. Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more. A view hierarchy is placed within an activity's window by the {@link android.app.Activity#setContentView Activity.setContentView()} method. The *content view* is

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.) **Services** A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the {@link android.app.Service} base class. A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback. Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback). See Processes and Threads, later. All the activities in a task move together as a unit. The entire task (the entire activity stack) can be brought to the foreground or sent to the background. Suppose, for instance, that the current task has four activities in its stack — three under the current activity. The user presses the HOME key, goes to the application launcher, and selects a new application (actually, a new *task*). The current task goes into the background and the root activity for the new task is displayed. Then, after a short period, the user goes back to the home screen and again selects the previous application (the previous task). That task, with all four activities in the stack, comes forward. When the user presses the BACK key, the screen does not

'184 Claims	**Android Device with One or More Apps**
	display the activity the user just left (the root activity of the previous task). Rather, the activity on the top of the stack is removed and the previous activity in the same task is displayed. As noted above, there's never more than one instance of a "{@code singleTask}" or "{@code singleInstance}" activity, so that instance is expected to handle all new intents. A "{@code singleInstance}" activity is always at the top of the stack (since it is the only activity in the task), so it is always in position to handle the intent. However, a "{@code singleTask}" activity may or may not have other activities above it in the stack. If it does, it is not in position to handle the intent, and the intent is dropped. (Even though the intent is dropped, its arrival would have caused the task to come to the foreground, where it would remain.) An activity has essentially three states: • It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions. • It is *paused* if it has lost focus but is still visible to the user. That is, another activity lies on top of it and that activity either is transparent or doesn't cover the full screen, so some of the paused activity can show through. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations. • It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere. Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them: • The **entire lifetime** of an activity happens between the first call to {@link android.app.Activity#onCreate onCreate()} through to a single final call to {@link android.app.Activity#onDestroy}. An activity does all its initial setup of "global" state in {@code onCreate()}, and releases all remaining resources in {@code onDestroy()}. For example, if it has

'184 Claims	Android Device with One or More Apps
	a thread running in the background to download data from the network, it may create that thread in {@code onCreate()} and then stop the thread in {@code onDestroy()}. • The **visible lifetime** of an activity happens between a call to {@link android.app.Activity#onStart onStart()} until a corresponding call to {@link android.app.Activity#onStop onStop()}. During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. For example, you can register a {@link android.content.BroadcastReceiver} in {@code onStart()} to monitor for changes that impact your UI, and unregister it in {@code onStop()} when the user can no longer see what you are displaying. The {@code onStart()} and {@code onStop()} methods can be called multiple times, as the activity alternates between being visible and hidden to the user. • The **foreground lifetime** of an activity happens between a call to {@link android.app.Activity#onResume onResume()} until a corresponding call to {@link android.app.Activity#onPause onPause()}. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the resumed and paused states — for example, {@code onPause()} is called when the device goes to sleep or when a new activity is started, {@code onResume()} is called when an activity result or a new intent is delivered. Therefore, the code in these two methods should be fairly lightweight. {@link android.app.Activity#onStart onStart()} — Called just before the activity becomes visible to the user. Followed by {@code onResume()} if the activity comes to the foreground, or {@code onStop()} if it becomes hidden. — No — {@code onResume()} or {@code onStop()} Processes and lifecycles The Android system tries to maintain an application process for as long as possible, but eventually it will need to remove old processes when memory runs low. To determine which processes to keep and which to kill, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components. Processes with the lowest importance are eliminated first, then those

'184 Claims	Android Device with One or More Apps
	with the next lowest, and so on. There are five levels in the hierarchy. The following list presents them in order of importance: 1. A **foreground process** is one that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions hold: ○ It is running an activity that the user is interacting with (the Activity object's {@link android.app.Activity#onResume onResume()} method has been called). ○ It hosts a service that's bound to the activity that the user is interacting with. ○ It has a {@link android.app.Service} object that's executing one of its lifecycle callbacks ({@link android.app.Service#onCreate onCreate()}, {@link android.app.Service#onStart onStart()}, or {@link android.app.Service#onDestroy onDestroy()}). ○ It has a {@link android.content.BroadcastReceiver} object that's executing its {@link android.content.BroadcastReceiver#onReceive onReceive()} method. Only a few foreground processes will exist at any given time. They are killed only as a last resort — if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive. 2. A **visible process** is one that doesn't have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions holds: ○ It hosts an activity that is not in the foreground, but is still visible to the user (its {@link android.app.Activity#onPause onPause()} method has been called). This may occur, for example, if the foreground activity is a dialog that allows the previous activity to be seen behind it. ○ It hosts a service that's bound to a visible activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. 3. A **service process** is one that is running a service that has been started with the {@link android.content.Context#startService startService()} method and that does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing an mp3 in the

'184 Claims	Android Device with One or More Apps
	background or downloading data on the network), so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes. 4. A **background process** is one holding an activity that's not currently visible to the user (the Activity object's {@link android.app.Activity#onStop onStop()} method has been called). These processes have no direct impact on the user experience, and can be killed at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and captures its current state, killing its process will not have a deleterious effect on the user experience. 5. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. **SAMSUNG_PRIORART0005487, Activity** * An activity has essentially four states: * * If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * If an activity has lost focus but is still visible (that is, a new non-full-sized * or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme * low memory situations. * If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be * completely restarted and restored to its previous state. * - The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user. * * - The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new * intent is delivered -- so the code in these methods should be fairly * lightweight. *

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* Here is an excerpt from a calendar activity that stores the user's * preferred view mode in its persistent settings: * * <pre class="prettyprint"> * public class CalendarActivity extends Activity { * ... * * static final int DAY_VIEW_MODE = 0; * static final int WEEK_VIEW_MODE = 1; * The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has * reached a memory paging state, so this is required in order to keep the user * interface responsive. * A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * A background activity (an activity that is not visible to * the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its * {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it. * An empty process is one hosting no activities or other

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * * <p>Sometimes an Activity may need to do a long-running operation that exists * independently of the activity lifecycle itself. An example may be a camera * application that allows you to upload a picture to a web site. The upload * may take a long time, and the application should allow the user to leave * the application will it is executing. To accomplish this, your Activity * should start a {@link Service} in which the upload takes place. This allows * the system to properly prioritize your process (considering it to be more * important than other non-visible applications) for the duration of the * upload, independent of whether the original activity is paused, stopped, * or finished. */ **SAMSUNG_PRIORART0005350, ActivityManager** /** * Return a list of the tasks that the user has recently launched, with * the most recent being first and older ones after in order. * * @param maxNum The maximum number of entries to return in the list. The * actual number returned may be smaller, depending on how many tasks the * user has started and the maximum number the system can remember. *

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* @return Returns a list of RecentTaskInfo records describing each of
	* the recent tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RecentTaskInfo> getRecentTasks(int maxNum, int flags)
	throws SecurityException {
	try {
	return ActivityManagerNative.getDefault().getRecentTasks(maxNum,
	flags);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Return a list of the tasks that are currently running, with
	* the most recent being first and older ones after in order. Note that
	* "running" does not mean any of the task's code is currently loaded or
	* activity -- the task may have been frozen by the system, so that it
	* can be restarted in its previous state when next brought to the
	* foreground.
	*
	* @param maxNum The maximum number of entries to return in the list. The
	* actual number returned may be smaller, depending on how many tasks the
	* user has started.
	*
	* @return Returns a list of RunningTaskInfo records describing each of

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* the running tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RunningTaskInfo> getRunningTasks(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningTaskInfo>)ActivityManagerNative.getDefault()
	.getTasks(maxNum, 0, null);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Set to true if the service has asked to run as a foreground process.
	*/
	public boolean foreground;
	
	/**
	* Return a list of the services that are currently running.
	*
	* @param maxNum The maximum number of entries to return in the list. The
	* actual number returned may be smaller, depending on how many services
	* are running.
	*
	* @return Returns a list of RunningServiceInfo records describing each of

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* the running tasks.
	*/
	public List<RunningServiceInfo> getRunningServices(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningServiceInfo>)ActivityManagerNative.getDefault()
	.getServices(maxNum, 0);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Constant for {@link #importance}: this process is running the
	* foreground UI.
	*/
	public static final int IMPORTANCE_FOREGROUND = 100;
	
	/**
	* Constant for {@link #importance}: this process is running something
	* that is considered to be actively visible to the user.
	*/
	public static final int IMPORTANCE_VISIBLE = 200;
	
	/**
	* Constant for {@link #importance}: this process is contains services
	* that should remain running.
	*/
	public static final int IMPORTANCE_SERVICE = 300;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* Constant for {@link #importance}: this process process contains
	* background code that is expendable.
	*/
	public static final int IMPORTANCE_BACKGROUND = 400;
	
	/**
	* The relative importance level that the system places on this
	* process. May be one of {@link #IMPORTANCE_FOREGROUND},
	* {@link #IMPORTANCE_VISIBLE}, {@link #IMPORTANCE_SERVICE},
	* {@link #IMPORTANCE_BACKGROUND}, or {@link #IMPORTANCE_EMPTY}. These
	* constants are numbered so that "more important" values are always
	* smaller than "less important" values.
	*/
	public int importance;
	
	/**
	* An additional ordering within a particular {@link #importance}
	* category, providing finer-grained information about the relative
	* utility of processes within a category. This number means nothing
	* except that a smaller values are more recently used (and thus
	* more important). Currently an LRU value is only maintained for
	* the {@link #IMPORTANCE_BACKGROUND} category, though others may
	* be maintained in the future.
	*/
	public int lru;
	
	public RunningAppProcessInfo() {
	importance = IMPORTANCE_FOREGROUND;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	
	/**
	* Returns a list of application processes that are running on the device.
	*
	* @return Returns a list of RunningAppProcessInfo records, or null if there are no
	* running processes (it will not return an empty list). This list ordering is not
	* specified.
	*/
	public List<RunningAppProcessInfo> getRunningAppProcesses() {
	try {
	return ActivityManagerNative.getDefault().getRunningAppProcesses();
	} catch (RemoteException e) {
	return null;
	}
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityManager
	
	/**
	* Class that answers queries about the state of network connectivity. It also
	* notifies applications when network connectivity changes. Get an instance
	* of this class by calling
	* {@link android.content.Context#getSystemService(String)
	Context.getSystemService(Context.CONNECTIVITY_SERVICE)}.
	* <p>
	* The primary responsibilities of this class are to:
	*
	* - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
	* - Send broadcast intents when network connectivity changes

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* - Attempt to "fail over" to another network when connectivity to a network
	* is lost
	* - Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	@SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
	public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED =
	"android.net.conn.BACKGROUND_DATA_SETTING_CHANGED";
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	public void setNetworkPreference(int preference) {
	try {
	mService.setNetworkPreference(preference);
	} catch (RemoteException e) {
	}
	}
	
	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {
	return false;
	}
	}
	
	/**
	* Returns the value of the setting for background data usage. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing
	* a connection to the preferred network is taken care of when we handle
	* the disconnect event from the non-preferred network
	* (see {@link #handleDisconnect(NetworkInfo)}).
	*/
	private void enforcePreference() {
	if (mActiveNetwork == null)
	return;
	
	for (NetworkStateTracker t : mNetTrackers) {
	if (t == mActiveNetwork) {
	int netType = t.getNetworkInfo().getType();
	int otherNetType = ((netType == ConnectivityManager.TYPE_WIFI) ?
	ConnectivityManager.TYPE_MOBILE :
	ConnectivityManager.TYPE_WIFI);
	
	if (t.getNetworkInfo().getType() != mNetworkPreference) {
	NetworkStateTracker otherTracker = mNetTrackers[otherNetType];
	if (otherTracker.isAvailable()) {
	teardown(t);
	}
	}
	}
	}
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* @see ConnectivityManager#getBackgroundDataSetting()
	*/
	public boolean getBackgroundDataSetting() {
	return Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, 1) == 1;
	}
	
	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {
	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}
	
	/**
	* See if the other network is available to fail over to.
	* If is not available, we enable it anyway, so that it

'184 Claims	Android Device with One or More Apps

	* will be able to connect when it does become available,
	* but we report a total loss of connectivity rather than
	* report that we are attempting to fail over.
	*/
	NetworkInfo switchTo = null;
	if (newNet.isAvailable()) {
	mActiveNetwork = newNet;
	switchTo = newNet.getNetworkInfo();
	switchTo.setFailover(true);
	if (!switchTo.isConnectedOrConnecting()) {
	newNet.reconnect();
	}
	} else {
	newNet.reconnect();
	}
	
	if (info.getType() == ConnectivityManager.TYPE_MOBILE) {
	otherNet = mWifiStateTracker;
	} else /* info().getType() == TYPE_WIFI */ {
	otherNet = mMobileDataStateTracker;
	}
	
	int incrValue = ConnectivityManager.TYPE_MOBILE - ConnectivityManager.TYPE_WIFI;
	int stopValue = ConnectivityManager.TYPE_MOBILE + incrValue;
	
	Android 2.2
	
	SAMSUNG_PRIORART0005350, Fundamentals.jd

'184 Claims	Android Device with One or More Apps
	Activities An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others. Each one is implemented as a subclass of the {@link android.app.Activity} base class. An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one. Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen. The visual content of the window is provided by a hierarchy of views — objects derived from the base {@link android.view.View} class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space. Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more. A view hierarchy is placed within an activity's window by the {@link android.app.Activity#setContentView Activity.setContentView()} method. The *content view* is the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.)

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Services A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the {@link android.app.Service} base class. A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback. Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback). See Processes and Threads, later. All the activities in a task move together as a unit. The entire task (the entire activity stack) can be brought to the foreground or sent to the background. Suppose, for instance, that the current task has four activities in its stack — three under the current activity. The user presses the HOME key, goes to the application launcher, and selects a new application (actually, a new *task*). The current task goes into the background and the root activity for the new task is displayed. Then, after a short period, the user goes back to the home screen and again selects the previous application (the previous task). That task, with all four activities in the stack, comes forward. When the user presses the BACK key, the screen does not display the activity the user just left (the root activity of the previous task). Rather, the activity on the top of the stack is removed and the previous activity in the same task is displayed.

'184 Claims	**Android Device with One or More Apps**
	As noted above, there's never more than one instance of a "{@code singleTask}" or "{@code singleInstance}" activity, so that instance is expected to handle all new intents. A "{@code singleInstance}" activity is always at the top of the stack (since it is the only activity in the task), so it is always in position to handle the intent. However, a "{@code singleTask}" activity may or may not have other activities above it in the stack. If it does, it is not in position to handle the intent, and the intent is dropped. (Even though the intent is dropped, its arrival would have caused the task to come to the foreground, where it would remain.) An activity has essentially three states: • It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions. • It is *paused* if it has lost focus but is still visible to the user. That is, another activity lies on top of it and that activity either is transparent or doesn't cover the full screen, so some of the paused activity can show through. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations. • It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere. Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them: • The **entire lifetime** of an activity happens between the first call to {@link android.app.Activity#onCreate onCreate()} through to a single final call to {@link android.app.Activity#onDestroy}. An activity does all its initial setup of "global" state in {@code onCreate()}, and releases all remaining resources in {@code onDestroy()}. For example, if it has a thread running in the background to download data from the network, it may create that thread in {@code onCreate()} and then stop the thread in {@code onDestroy()}. • The **visible lifetime** of an activity happens between a call to {@link android.app.Activity#onStart onStart()} until a corresponding call to {@link android.app.Activity#onStop onStop()}. During

'184 Claims	Android Device with One or More Apps
	this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. For example, you can register a {@link android.content.BroadcastReceiver} in {@code onStart()} to monitor for changes that impact your UI, and unregister it in {@code onStop()} when the user can no longer see what you are displaying. The {@code onStart()} and {@code onStop()} methods can be called multiple times, as the activity alternates between being visible and hidden to the user. • The **foreground lifetime** of an activity happens between a call to {@link android.app.Activity#onResume onResume()} until a corresponding call to {@link android.app.Activity#onPause onPause()}. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the resumed and paused states — for example, {@code onPause()} is called when the device goes to sleep or when a new activity is started, {@code onResume()} is called when an activity result or a new intent is delivered. Therefore, the code in these two methods should be fairly lightweight. `{@link android.app.Activity#onStart onStart()}` — Called just before the activity becomes visible to the user. Followed by {@code onResume()} if the activity comes to the foreground, or {@code onStop()} if it becomes hidden. — No — {@code onResume()} or {@code onStop()} Processes and lifecycles The Android system tries to maintain an application process for as long as possible, but eventually it will need to remove old processes when memory runs low. To determine which processes to keep and which to kill, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components. Processes with the lowest importance are eliminated first, then those with the next lowest, and so on. There are five levels in the hierarchy. The following list presents them in order of importance: 6. A **foreground process** is one that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions hold:

'184 Claims	Android Device with One or More Apps
	○ It is running an activity that the user is interacting with (the Activity object's {@link android.app.Activity#onResume onResume()} method has been called). ○ It hosts a service that's bound to the activity that the user is interacting with. ○ It has a {@link android.app.Service} object that's executing one of its lifecycle callbacks ({@link android.app.Service#onCreate onCreate()}, {@link android.app.Service#onStart onStart()}, or {@link android.app.Service#onDestroy onDestroy()}). ○ It has a {@link android.content.BroadcastReceiver} object that's executing its {@link android.content.BroadcastReceiver#onReceive onReceive()} method. Only a few foreground processes will exist at any given time. They are killed only as a last resort — if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive. 7. A **visible process** is one that doesn't have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions holds: ○ It hosts an activity that is not in the foreground, but is still visible to the user (its {@link android.app.Activity#onPause onPause()} method has been called). This may occur, for example, if the foreground activity is a dialog that allows the previous activity to be seen behind it. ○ It hosts a service that's bound to a visible activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. 8. A **service process** is one that is running a service that has been started with the {@link android.content.Context#startService startService()} method and that does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing an mp3 in the background or downloading data on the network), so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes. 9. A **background process** is one holding an activity that's not currently visible to the user (the Activity object's {@link android.app.Activity#onStop onStop()} method has been called). These

'184 Claims	Android Device with One or More Apps
	processes have no direct impact on the user experience, and can be killed at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and captures its current state, killing its process will not have a deleterious effect on the user experience. 10. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. **SAMSUNG_PRIORART0005487, Activity** * Activities in the system are managed as an activity stack. * When a new activity is started, it is placed on the top of the stack * and becomes the running activity -- the previous activity always remains * below it in the stack, and will not come to the foreground again until * the new activity exits. * * An activity has essentially four states: * * If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * If an activity has lost focus but is still visible (that is, a new non-full-sized * or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* low memory situations. * - If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden * and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be * completely restarted and restored to its previous state. * * * The following diagram shows the important state paths of an Activity. * The square rectangles represent callback methods you can implement to * perform operations when the Activity moves between states. The colored * ovals are major states the Activity can be in. * * * * There are three key loops you may be interested in monitoring within your * activity: * * * - The entire lifetime of an activity happens between the first call * to {@link android.app.Activity#onCreate} through to a single final call * to {@link android.app.Activity#onDestroy}. An activity will do all setup * of "global" state in onCreate(), and release all remaining resources in * onDestroy(). For example, if it has a thread running in the background * to download data from the network, it may create that thread in onCreate()

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* and then stop the thread in onDestroy(). * * The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user. * * The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new * intent is delivered -- so the code in these methods should be fairly * lightweight. * * * <tr><th colspan="2" align="left" border="0">{@link android.app.Activity#onStart onStart()}</th> * <td>Called when the activity is becoming visible to the user. * <p>Followed by <code>onResume()</code> if the activity comes * to the foreground, or <code>onStop()</code> if it becomes hidden.</td>

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* <td align="center">No</td> * <td align="center"><code>onResume()</code> or <code>onStop()</code></td> * </tr> * * <tr><td rowspan="2" style="border-left: none;"> </td> * <th align="left" border="0">{@link android.app.Activity#onResume onResume()}</th> * <td>Called when the activity will start * interacting with the user. At this point your activity is at * the top of the activity stack, with user input going to it. * Always followed by <code>onPause()</code>.</td> * <td align="center">No</td> * <td align="center"><code>onPause()</code></td> * </tr> * * * Unless you specify otherwise, a configuration change (such as a change * in screen orientation, language, input devices, etc) will cause your * current activity to be destroyed, going through the normal activity * lifecycle process of {@link #onPause}, * {@link #onStop}, and {@link #onDestroy} as appropriate. If the activity * had been in the foreground or visible to the user, once {@link #onDestroy} is * called in that instance then a new instance of the activity will be * created, with whatever savedInstanceState the previous instance had generated * from {@link #onSaveInstanceState}. * The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has

'184 Claims	Android Device with One or More Apps
	* reached a memory paging state, so this is required in order to keep the user * interface responsive. * <p>A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * <p>A background activity (an activity that is not visible to * the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its * {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it. * <p>An empty process is one hosting no activities or other * application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * * <p>Sometimes an Activity may need to do a long-running operation that exists * independently of the activity lifecycle itself. An example may be a camera * application that allows you to upload a picture to a web site. The upload * may take a long time, and the application should allow the user to leave * the application will it is executing. To accomplish this, your Activity * should start a {@link Service} in which the upload takes place. This allows * the system to properly prioritize your process (considering it to be more

'184 Claims	Android Device with One or More Apps

	* important than other non-visible applications) for the duration of the
	* upload, independent of whether the original activity is paused, stopped,
	* or finished.
	*/
	
	/**
	* Called as part of the activity lifecycle when an activity is about to go
	* into the background as the result of user choice. For example, when the
	* user presses the Home key, {@link #onUserLeaveHint} will be called, but
	* when an incoming phone call causes the in-call Activity to be automatically
	* brought to the foreground, {@link #onUserLeaveHint} will not be called on
	* the activity being interrupted. In cases when it is invoked, this method
	* is called right before the activity's {@link #onPause} callback.
	*
	* <p>This callback and {@link #onUserInteraction} are intended to help
	* activities manage status bar notifications intelligently; specifically,
	* for helping activities determine the proper time to cancel a notfication.
	*
	* @see #onUserInteraction()
	*/
	protected void onUserLeaveHint() {
	}
	
	* <p>As a general rule, however, a resumed activity will have window
	* focus... unless it has displayed other dialogs or popups that take
	* input focus, in which case the activity itself will not have focus
	* when the other windows have it. Likewise, the system may display
	* system-level windows (such as the status bar notification panel or
	* a system alert) which will temporarily take window input focus without
	* pausing the foreground activity.

'184 Claims	Android Device with One or More Apps
	* * @param hasFocus Whether the window of this activity has focus. * * @see #hasWindowFocus() /** * Change the desired orientation of this activity. If the activity * is currently in the foreground or otherwise impacting the screen * orientation, the screen will immediately be changed (possibly causing * the activity to be restarted). Otherwise, this will be used the next * time the activity is visible. * * @param requestedOrientation An orientation constant as used in * {@link ActivityInfo#screenOrientation ActivityInfo.screenOrientation}. */ public void setRequestedOrientation(int requestedOrientation) { **GOOG-HEADWATER-00000029, SAMSUNG_PRIORART0005353, ConnectivityManager** /** * Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling * {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network * is lost * - Provide an API that allows applications to query the coarse-grained or fine-grained * state of the available networks * */ * A change in network connectivity has occurred. A connection has either * been established or lost. The NetworkInfo for the affected network is * sent as an extra; it should be consulted to see what kind of * connectivity event occurred. /** * Broadcast Action: The setting for **background data usage** has changed * values. Use {@link #getBackgroundDataSetting()} to get the current value. * <p> * If an application uses the network in the background, it should listen * for this broadcast and stop using the background data if the value is * false. */ @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION) public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED = "android.net.conn.BACKGROUND_DATA_SETTING_CHANGED"; /** * The Default Mobile data connection. When active, all data traffic * will use this connection by default. Should not coexist with other

'184 Claims	Android Device with One or More Apps

	* default connections.
	*/
	public static final int TYPE_MOBILE = 0;
	/**
	* The Default WIFI data connection. When active, all data traffic
	* will use this connection by default. Should not coexist with other
	* default connections.
	*/
	public static final int TYPE_WIFI = 1;
	
	/**
	* Returns the value of the setting **for background data usage**. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	/**
	* Sets the persisted value for enabling/disabling Mobile data.
	*
	* @param enabled Whether the mobile data connection should be
	* used or not.
	* @hide
	*/
	public void setMobileDataEnabled(boolean enabled) {
	try {
	mService.setMobileDataEnabled(enabled);

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	} catch (RemoteException e) {
	}
	}
	See also Android Developers Blog_Multitasking the Android Way, GOOG-HEADWATER-00000025-27
	
	A key to how Android handles applications in this way is that processes don't shut down cleanly. When the user leaves an application, its process is kept around in the background, allowing it to continue working (for example downloading web pages) if needed, and come immediately to the foreground if the user returns to it. If a device never runs out of memory, then Android will keep all of these processes around, truly leaving all applications "running" all of the time.
	
	Explicitly running in the background
	
	So far, we have a way for applications to implicitly do work in the background, as long as the process doesn't get killed by Android as part of its regular memory management. This is fine for things like loading web pages in the background, but what about features with harder requirements? Background music playback, data synchronization, location tracking, alarm clocks, etc.
	
	For these tasks, the application needs a way to tell Android "I would explicitly like to run at this point." There are two main facilities available to applications for this, represented by two kinds of components they can publish in their manifest: *broadcast receivers* and *services*.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Broadcast Receivers A BroadcastReceiver allows an application to run, for a brief amount of time, in the background as a result of something else happening. It can be used in many ways to build higher-level facilities: for example the AlarmManager allows an application to have a broadcast sent at a certain time in the future, and the LocationManager can send a broadcast when it detects interesting changes in location. Because information about the receiver is part of an application's manifest, Android can find and launch the application even if it isn't running; of course if it already has its process available in the background, the broadcast can very efficiently be directly dispatched to it. When handling a broadcast, the application is given a fixed set of time (currently 10 seconds) in which to do its work. If it doesn't complete in that time, the application is considered to be misbehaving, and its process immediately tossed into the background state to be killed for memory if needed. Broadcast receivers are great for doing small pieces of work in response to an external stimulus, such as posting a notification to the user after being sent a new GPS location report. They are very lightweight, since the application's process only needs to be around while actively receiving the broadcast. Because they are active for a deterministic amount of time, fairly strong guarantees can be made about not killing their process while running. However they are not appropriate for anything of indeterminate length, such as networking.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Services A Service allows an application to implement longer-running background operations. There are actually a lot of other functions that services provide, but for the discussion here their fundamental purpose is for an application to say "hey I would like to continue running even while in the background, until I say I am done." An application controls when its service runs by explicitly starting and stopping the service. While services do provide a rich client-server model, its use is optional. Upon starting an application's services, Android simply instantiates the component in the application's process to provide its context. How it is used after that is up to the application: it can put all of the needed code inside of the service itself without interacting with other parts of the application, make calls on other singleton objects shared with other parts of the app, directly retrieve the Service instance from elsewhere if needed, or run it in another process and do a full-blown RPC protocol if that is desired. Process management for services is different than broadcast receivers, because an unbounded number of services can ask to be running for an unknown amount of time. There may not be enough RAM to have all of the requesting services run, so as a result no strong guarantees are made about being able to keep them running. If there is too little RAM, processes hosting services will be immediately killed like background processes are. However, if appropriate, Android will remember that these services wish to remain running, and restart their process at a later time when more RAM is available. For example, if the user goes to a web page that requires large amounts of RAM, Android may kill background service processes like sync until the browser's memory needs go down. Services can further negotiate this behavior by requesting they be considered "foreground." This places the service in a "please don't kill" state, but requires that it include a notification to the user about it actively running. This is useful for services such as background music playback or car navigation, which the user is actively aware of; when you're playing music and using the browser, you can always see the music-playing glyph in the status bar. Android won't try to kill these services, but as a trade-off, ensures the user knows about them and is able to explicitly stop them when desired.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	The value of generic components Android's generic broadcast receiver and service components allow developers to create a wide variety of efficient background operations, including things that were never originally considered. In Android 1.0 they were used to implement nearly all of the background behavior that the built-in and proprietary Google apps provided: • Music playback runs in a service to allow it to continue operating after the user leaves the music application. • The alarm clock schedules a broadcast receiver with the alarm manager, to go off at the next set alarm time.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	• The calendar application likewise schedules an alarm to display or update its notification at the appropriate time for the next calendar event. • Background file download is implemented a service that runs when there are any downloads to process. • The e-mail application schedules an alarm to wake up a service at regular intervals that looks for and retrieves any new mail. • The Google applications maintain a service to receive push notifications from the network; it in turn sends broadcasts to individual apps when it is told that they need to do things like synchronize contacts. As the platform has evolved, these same basic components have been used to implement many of the major new developer features: • Input methods are implemented by developers as a Service component that Android manages and works with to display as the current IME. • Application widgets are broadcast receivers that Android sends broadcasts to when it needs to interact with them. This allows app widgets to be quite lightweight, by not needing their application's process remain running. • Accessibility features are implemented as services that Android keeps running while in use and sends appropriate information to about user interactions. • Sync adapters introduced in Android 2.0 are services that are run in the background when a particular data sync needs to be performed. • Live wallpapers are a service started by Android when selected by the user.
[1d] apply a differential traffic control policy to the Internet service access request, based on (i) whether the application is classified as interacting with the	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: **Nexus One** *See, e.g.,* SAMSUNG_PRIORART0000001 (Nexus) at 320:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

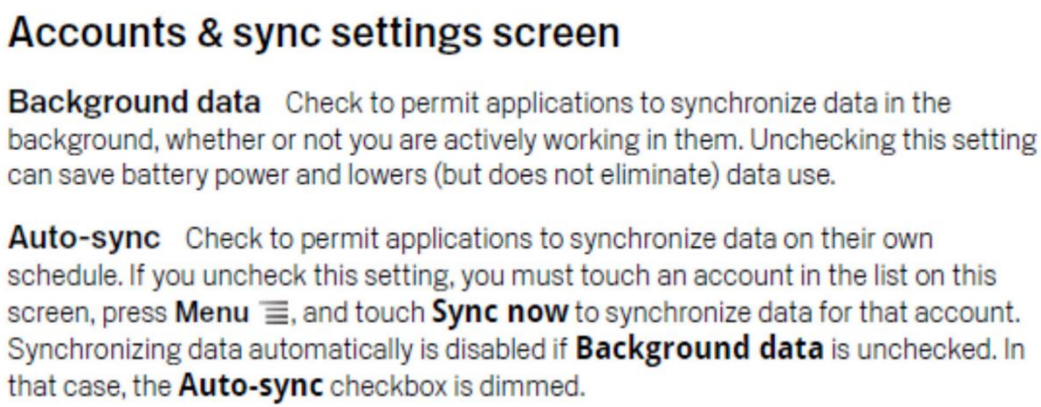
'184 Claims	Android Device with One or More Apps
user, and (ii) a differential traffic control policy list distinguishing between a first one or more applications resident on the device and a second one or more applications resident on the device, such that, the one or more processors are operable to, in a first state wherein the particular application is one of the first one or more applications, and the particular application is not classified as interacting with a user in the device user interface foreground, block the Internet service access request, and the one or more processors are operable to allow the Internet service access request in at least one other state, and	 SAMSUNG_PRIORART0000001 (Nexus) at 115-116 ("You can configure background data use and synchronization options for all of the applications on your phone. You can also configure what kinds of data you synchronize for each account. Some applications, such as Gmail and Calendar, have their own synchronization settings."). **HTC Dream / T-Mobile G1:** SAMSUNG_PRIORART0005598

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	  <u>JuiceDefender App</u> SAMSUNG_PRIORART0000379 (Latedroid):

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

JuiceDefender

JuiceDefender saves battery power (*lots of it!*) by controlling the device data connection and/or WiFi.

You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions ;)

The *Easy Mode* is a no-fuss one-click way to let your battery last longer - much longer. Just enable JuiceDefender by clicking on the big button and you're ready to go!

If you want more fine-grained control, try *Advanced Mode*, where you can configure all JuiceDefender features to your liking.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	There are 5 *triggers* for the enable/disable behaviour: **Battery** - when battery level gets low (less than 15%), disable APN/WiFi, and re-enable them when battery level is restored. APN/WiFi will also be enabled while the device is being recharged. **Schedule** - regularly enable APN/WiFi for a short period of time, to *let background data sync* occur (email, Twitter, Facebook, stock quotes...). If *Quick* is disabled APN/WiFi stays enabled for a longer period, useful if your data connection is very slow or you need to sync lots of data. **Night schedule** (requires *UltimateJuice*) - disable APN/WiFi during night time; you can also optionally put the phone in Silent Mode. **Screen** - enable APN/WiFi *while the screen is on* to allow browsing, tweeting, procrastination and general internet-powered enjoyment, regardless of scheduled events and battery level. **Location** (requires *UltimateJuice*) - this trigger controlled by the *'AutoWiFi'* button. It disables WiFi when the device is not in range of any known WiFi network. The location is determined via the cellular network, so it's usually quite coarse. It's a fully automatic set-it-and-forget-it WiFi manager! The *priority order* of the triggers is 1) location (WiFi only), 2) screen, 3) battery, 4) night schedule, 5) schedule - this means, for example, that when the screen is on APN/WiFi will be enabled even when the battery is low, or that the regular schedule won't occur during the night period. SAMSUNG_PRIORART0000361 (Purdy):

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

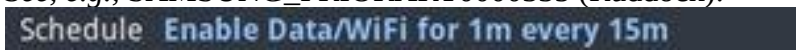
'184 Claims	Android Device with One or More Apps

Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power.

The whole point of a smartphone with Google apps baked in is constant connectivity, of course, and you don't want to shut off access to your email, Google Voice messages, and other online services. But when you're walking around, at your office desk, and generally not actively using your phone, you probably don't need your phone to check in every minute with the mothership. JuiceDefender lets you set a time interval—5 minutes, 15, 30, an hour, two hours—at which its background process will re-enable your carrier APN, see if there are new messages or data coming in, and then shut off again. You can also set similar Wi-Fi connectivity rules, or only have web data enabled when you've got your screen on. Besides the battery savings, those who like to parse out their email checks and avoid minute-by-minute distractions see some benefit here, too.

See, e.g., SAMSUNG_PRIORART0000351 (Configuration-Translated):

"• **Schedule:** used to configure how often Juice Defender will activate the APN / WIFI to synchronize emails, social networks, etc .., we select the interval in the bar of your submenu for example 5 minutes every half hour also has the **Quick** button to minimize the connection time for the previous example

'184 Claims	Android Device with One or More Apps
	would be 2 minutes every half hour, I recommend not using this option if the selected time period is too short, it may not give you time to connect." *See, e.g.*, SAMSUNG_PRIORART0000361 (Purdy): "JuiceDefender lets you set a time interval—5 minutes, 15, 30, an hour, two hours— at which its background process will re-enable your carrier APN, see if there are new messages or data coming in, and then shut off again." *See, e.g.*, SAMSUNG_PRIORART0000335 (Ruddock):  **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Manage Wifi If this setting is selected, then Green Power will regularly turn on and off the Wifi connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Wifi as it is, never turning it on or off. Please note that if you manually switches off the Wifi, then Green Power will unselect the "Manage Wifi" setting in order not to automatically switch on the Wifi again despite your manual action. Then, if you switch back on the Wifi or reselect "Manage Wifi" setting, Green Power will resume managing Wifi connection. Manage Mobile Network

'184 Claims	Android Device with One or More Apps
	If this setting is selected, then Green Power will regularly turn on and off the Mobile Network connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Mobile Network as it is, never turning it on or off. Please note that in order for Green Power to turn on / off Mobile Network, this one has to be manually enabled by the user first in the phone settings (Wireless & networks → Mobile Network) or in Green Power settings (Global wireless settings → Mobile Network) . Green Power can't itself turn on Mobile Network as this is a limitation of the Android system for security and cost reasons."). *See also, e.g., [1b], [1c].*
[1e] for at least one of the first or second one or more applications, disallow Internet data communication for that application based on an application-specific amount of Internet data usage reaching a limit.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: **Nexus One** *See, e.g.,* SAMSUNG_PRIORART0000001 (Nexus) at 67: **To limit your data connection to 2G networks** You can extend your battery life by limiting your data connections to 2G networks (GPRS or EDGE). When you are connected to a 2G network, you may want to postpone activities that transmit a lot of data, such as sending, uploading, or downloading pictures or video, until you are connected to a faster mobile or other wireless network. 1 Press **Home**, press **Menu**, and touch **Settings** to open the Settings application. 2 Touch **Wireless & networks** › **Mobile networks** and check **Use only 2G networks**.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	See, e.g., SAMSUNG_PRIORART0000001 (Nexus) at 199: **Exchanging messages** You can send text (SMS) messages of up to 160 characters to another mobile phone. If you keep typing after the limit, your message is delivered as a series of messages. Multimedia (MMS) messages can contain text and a picture, a recorded voice, an audio file, a video, or a picture slideshow. Messaging automatically converts a message into a multimedia message if you send it to an email address instead of a phone number, add a subject, or attach a media file. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 203: **Multimedia message (MMS) settings** **Delivery reports** Check to receive a report on the messages you send. **Read reports** Check to receive reports when your messages are read or deleted. **Auto-retrieve** Uncheck to download only the heading of multimedia messages, which you can then touch to open a menu to download the whole message. This is useful for controlling how much data you download, for example, when connected to a slow data network. **Roaming auto-retrieve** Uncheck to download only the header of multimedia messages when connected to other carrier's data networks. This is useful for avoiding unexpected charges, if your contract has limits on data roaming. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 260:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	To share a video You can share a video by sending it in an email or with a multimedia (MMS) message. MMS messages have size limits, typically 3MB, or approximately 1 minute of high-quality video or 2 minutes of low-quality video. See "Changing Camera settings" on page 247. You can also share a video by uploading it to the YouTube web site. 1 While viewing an album, press **Menu** twice. 2 Check the videos or pictures to share. 3 Touch **Share**. 4 In the menu that opens, touch the application to use to share the selected videos. The application you selected opens with the video attached to a new message. If you selected YouTube, you're prompted to enter details about the video to upload. For more information, see the section on that application. POUZERATE0000196 (GreenPower User Guide) ("Check Traffic If this is selected, then prior to turning off wireless, Green Power will check that there is no network traffic. If there is, it will wait a few seconds and checks again until there is no traffic anymore."). *See also, e.g.*, [1b], [1c]. For example: **SAMSUNG_PRIORART0005353, ThrottleManager** /** * Class that handles throttling. It provides read/write numbers per interface * and methods to apply throttled rates. * {@hide} */ /** * returns a long of the byte count either read or written on the named interface

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* for the period described. Direction is either DIRECTION_RX or DIRECTION_TX and
	* period may only be PERIOD_CYCLE for the current cycle (other periods may be supported
	* in the future). Ago indicates the number of periods in the past to lookup - 0 means
	* the current period, 1 is the last one, 2 was two periods ago..
	* {@hide}
	*/
	public long getByteCount(String iface, int direction, int period, int ago) {
	try {
	return mService.getByteCount(iface, direction, period, ago);
	} catch (RemoteException e) {
	return -1;
	}
	}
	
	/**
	* returns the number of bytes read+written after which a particular cliff
	* takes effect on the named iface. Currently only cliff #1 is supported (1 step)
	* {@hide}
	*/
	public long getCliffThreshold(String iface, int cliff) {
	try {
	return mService.getCliffThreshold(iface, cliff);
	} catch (RemoteException e) {
	return -1;
	}
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
[2] The wireless end-user device of claim 1, wherein the one or more processors are configured to classify that the particular application is interacting with the user in the device user interface foreground when the user of the device is directly interacting with that application or perceiving any benefit from that application.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 320: **Accounts & sync settings screen** **Background data** Check to permit applications to synchronize data in the background, whether or not you are actively working in them. Unchecking this setting can save battery power and lowers (but does not eliminate) data use. **Auto-sync** Check to permit applications to synchronize data on their own schedule. If you uncheck this setting, you must touch an account in the list on this screen, press **Menu**, and touch **Sync now** to synchronize data for that account. Synchronizing data automatically is disabled if **Background data** is unchecked. In that case, the **Auto-sync** checkbox is dimmed. SAMSUNG_PRIORART0000001 (Nexus) at 115-116 ("You can configure background data use and synchronization options for all of the applications on your phone. You can also configure what kinds of data you synchronize for each account. Some applications, such as Gmail and Calendar, have their own synchronization settings."). *See, e.g.*, the disclosures identified for claims [1c].

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
[3] The wireless end-user device of claim 1, wherein the one or more processors are configured to classify that the particular application is interacting with the user in the device user interface foreground based on a state of user interface priority for the application.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1c], [2].
[4] The wireless end-user device of claim 1, wherein the one or more processors are configured to classify that the particular application is not interacting with the user in the device user interface foreground when the application is providing or utilizing a background data service.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1c], [2].

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
[5] The wireless end-user device of claim 1, wherein the one or more processors are configured to, when the Internet service access request is blocked in the first state, delay network activity in association with the first Internet access request until a second Internet access request is received and allowed.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d].
[6] The wireless end-user device of claim 1, further comprising a user interface, wherein the user interface is to inform the user of the device when there are options to set, control, override, or modify at least one aspect of the differential traffic control policy and/or the differential traffic control policy list.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d].

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
[7] The wireless end-user device of claim 1, further comprising a wireless local area network (WLAN) modem to communicate data for Internet service activities between the device and at least one WLAN, when configured for and connected to the at least one WLAN, wherein the one or more processors are further configured to, for a time when the WLAN modem is to provide data communication for Internet service activities, not apply the differential traffic control policy.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d].
[8] The wireless end-user device of claim 7, wherein the one or more processors are further configured to, for a time	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d].

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
when the WLAN modem is to provide data communication for Internet service activities, not count an application-specific amount of Internet data usage toward the limit.	
[9] The wireless end-user device of claim 1, wherein the at least one WWAN comprises a home WWAN and a roaming WWAN, and wherein the one or more processors are further configured to apply the differential traffic control policy for a time when data communication for Internet service activities is provided by the WWAN modem and the roaming WWAN, and to not apply the differential traffic control policy for a time when data	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d].

'184 Claims	**Android Device with One or More Apps**
communication for Internet service activities is provided by the WWAN modem and the home WWAN.	
[10] The wireless end-user device of claim 1, wherein the one or more processors are further configured to dynamically change a determination of whether to apply the differential traffic control policy based on a power state of the device.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d]. In addition, *see, e.g.:* **Android 1.0** **SAMSUNG_PRIORART0005487, Power** /** * Wake lock that ensures that the CPU is running. The screen might * not be on. */ public static final int PARTIAL_WAKE_LOCK = 1; /** * Wake lock that ensures that the screen is on. */ public static final int FULL_WAKE_LOCK = 2; * Brightness value to use when battery is low */ public static final int BRIGHTNESS_LOW_BATTERY = 10;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	/**
	* Threshold for BRIGHTNESS_LOW_BATTERY (percentage)
	* Screen will stay dim if battery level is <= LOW_BATTERY_THRESHOLD
	*/
	public static final int LOW_BATTERY_THRESHOLD = 10;
	
	SAMSUNG_PRIORART0005487, BatteryManager
	
	// values for "status" field in the ACTION_BATTERY_CHANGED Intent
	public static final int BATTERY_STATUS_UNKNOWN = 1;
	public static final int BATTERY_STATUS_CHARGING = 2;
	public static final int BATTERY_STATUS_DISCHARGING = 3;
	public static final int BATTERY_STATUS_NOT_CHARGING = 4;
	public static final int BATTERY_STATUS_FULL = 5;
	
	// values for "health" field in the ACTION_BATTERY_CHANGED Intent
	public static final int BATTERY_HEALTH_UNKNOWN = 1;
	public static final int BATTERY_HEALTH_GOOD = 2;
	public static final int BATTERY_HEALTH_OVERHEAT = 3;
	public static final int BATTERY_HEALTH_DEAD = 4;
	public static final int BATTERY_HEALTH_OVER_VOLTAGE = 5;
	public static final int BATTERY_HEALTH_UNSPECIFIED_FAILURE = 6;
	
	// values of the "plugged" field in the ACTION_BATTERY_CHANGED intent
	public static final int BATTERY_PLUGGED_AC = 1;
	public static final int BATTERY_PLUGGED_USB = 2;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Android 1.6 **SAMSUNG_PRIORART0005350, Power** /** * Brightness value to use when battery is low */ public static final int BRIGHTNESS_LOW_BATTERY = 10; /** * Threshold for BRIGHTNESS_LOW_BATTERY (percentage) * Screen will stay dim if battery level is <= LOW_BATTERY_THRESHOLD */ public static final int LOW_BATTERY_THRESHOLD = 10; **SAMSUNG_PRIORART0005350, PowerManager** /** * This class gives you control of the power state of the device. * * <p>Device battery life will be significantly affected by the use of this API. Do not * acquire WakeLocks unless you really need them, use the minimum levels possible, and be sure * to release it as soon as you can. * * <p>You can obtain an instance of this class by calling * {@link android.content.Context#getSystemService(java.lang.String) Context.getSystemService()}. * * <p>The primary API you'll use is {@link #newWakeLock(int, String) newWakeLock()}. This will

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* create a {@link PowerManager.WakeLock} object. You can then use methods on this object to * control the power state of the device. In practice it's quite simple: * * {@samplecode * PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE); * PowerManager.WakeLock wl = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "My Tag"); * wl.acquire(); * ..screen will stay on during this section.. * wl.release(); * } * * <p>The following flags are defined, with varying effects on system power. These flags are * mutually exclusive - you may only specify one of them. * <table border="2" width="85%" align="center" frame="hsides" rules="rows"> * * <thead> * <tr><th>Flag Value</th> * <th>CPU</th> <th>Screen</th> <th>Keyboard</th></tr> * </thead> * * <tbody> * <tr><th>{@link #PARTIAL_WAKE_LOCK}</th> * <td>On*</td> <td>Off</td> <td>Off</td> * </tr> * * <tr><th>{@link #SCREEN_DIM_WAKE_LOCK}</th> * <td>On</td> <td>Dim</td> <td>Off</td> * </tr> *

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* <tr><th>{@link #SCREEN_BRIGHT_WAKE_LOCK}</th> * <td>On</td> <td>Bright</td> <td>Off</td> * </tr> * * <tr><th>{@link #FULL_WAKE_LOCK}</th> * <td>On</td> <td>Bright</td> <td>Bright</td> * </tr> * </tbody> * </table> * * <p>*If you hold a partial wakelock, the CPU will continue to run, irrespective of any timers * and even after the user presses the power button. In all other wakelocks, the CPU will run, but * the user can still put the device to sleep using the power button. * * <p>In addition, you can add two more flags, which affect behavior of the screen only. These * flags have no effect when combined with a {@link #PARTIAL_WAKE_LOCK}. * <table border="2" width="85%" align="center" frame="hsides" rules="rows"> * * <thead> * <tr><th>Flag Value</th> <th>Description</th></tr> * </thead> * * <tbody> * <tr><th>{@link #ACQUIRE_CAUSES_WAKEUP}</th> * <td>Normal wake locks don't actually turn on the illumination. Instead, they cause * the illumination to remain on once it turns on (e.g. from user activity). This flag * will force the screen and/or keyboard to turn on immediately, when the WakeLock is * acquired. A typical use would be for notifications which are important for the user to * see immediately.</td> * </tr>

'184 Claims	Android Device with One or More Apps

	*
	* <tr><th>{@link #ON_AFTER_RELEASE}</th>
	* <td>If this flag is set, the user activity timer will be reset when the WakeLock is
	* released, causing the illumination to remain on a bit longer. This can be used to
	* reduce flicker if you are cycling between wake lock conditions.</td>
	* </tr>
	* </tbody>
	* </table>
	*
	*
	*/
	
	private static final int WAKE_BIT_CPU_STRONG = 1;
	private static final int WAKE_BIT_CPU_WEAK = 2;
	private static final int WAKE_BIT_SCREEN_DIM = 4;
	private static final int WAKE_BIT_SCREEN_BRIGHT = 8;
	private static final int WAKE_BIT_KEYBOARD_BRIGHT = 16;
	
	SAMSUNG_PRIORART0005350, BatteryManager
	
	/**
	* The BatteryManager class contains strings and constants used for values
	* in the ACTION_BATTERY_CHANGED Intent.
	*/
	public class BatteryManager {
	
	// values for "status" field in the ACTION_BATTERY_CHANGED Intent
	public static final int BATTERY_STATUS_UNKNOWN = 1;
	public static final int BATTERY_STATUS_CHARGING = 2;

'184 Claims	Android Device with One or More Apps

	public static final int BATTERY_STATUS_DISCHARGING = 3;
	public static final int BATTERY_STATUS_NOT_CHARGING = 4;
	public static final int BATTERY_STATUS_FULL = 5;
	
	// values for "health" field in the ACTION_BATTERY_CHANGED Intent
	public static final int BATTERY_HEALTH_UNKNOWN = 1;
	public static final int BATTERY_HEALTH_GOOD = 2;
	public static final int BATTERY_HEALTH_OVERHEAT = 3;
	public static final int BATTERY_HEALTH_DEAD = 4;
	public static final int BATTERY_HEALTH_OVER_VOLTAGE = 5;
	public static final int BATTERY_HEALTH_UNSPECIFIED_FAILURE = 6;
	
	// values of the "plugged" field in the ACTION_BATTERY_CHANGED intent.
	// These must be powers of 2.
	/** Power source is an AC charger. */
	public static final int BATTERY_PLUGGED_AC = 1;
	/** Power source is a USB port. */
	public static final int BATTERY_PLUGGED_USB = 2;
	}
	
	SAMSUNG_PRIORART0005350, BatteryStats
	
	/**
	* A constant indicating a a wifi turn on timer
	*
	* {@hide}
	*/
	public static final int WIFI_TURNED_ON = 4;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	/** * A constant indicating a full wifi lock timer * * {@hide} */ public static final int FULL_WIFI_LOCK = 5; /** * A constant indicating a scan wifi lock timer * * {@hide} */ public static final int SCAN_WIFI_LOCK = 6; /** * A constant indicating a wifi multicast timer * * {@hide} */ public static final int WIFI_MULTICAST_ENABLED = 7; /** * A constant indicating an audio turn on timer * * {@hide} */ public static final int AUDIO_TURNED_ON = 7; /** * A constant indicating a video turn on timer

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* * {@hide} */ public static final int VIDEO_TURNED_ON = 8; /** * Include all of the data in the stats, including previously saved data. */ public static final int STATS_TOTAL = 0; /** * Include only the last run in the stats. */ public static final int STATS_LAST = 1; /** * Include only the current run in the stats. */ public static final int STATS_CURRENT = 2; /** * Include only the run since the last time the device was unplugged in the stats. */ public static final int STATS_UNPLUGGED = 3; public abstract void noteWifiTurnedOnLocked(); public abstract void noteWifiTurnedOffLocked(); public abstract void noteFullWifiLockAcquiredLocked(); public abstract void noteFullWifiLockReleasedLocked(); public abstract void noteScanWifiLockAcquiredLocked();

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	public abstract void noteScanWifiLockReleasedLocked(); public abstract void noteWifiMulticastEnabledLocked(); public abstract void noteWifiMulticastDisabledLocked(); /** * Returns the time in microseconds that the screen has been on while the device was * running on battery. * * {@hide} */ public abstract long getScreenOnTime(long batteryRealtime, int which); public static final int SCREEN_BRIGHTNESS_DARK = 0; public static final int SCREEN_BRIGHTNESS_DIM = 1; public static final int SCREEN_BRIGHTNESS_MEDIUM = 2; public static final int SCREEN_BRIGHTNESS_LIGHT = 3; public static final int SCREEN_BRIGHTNESS_BRIGHT = 4; public static final int DATA_CONNECTION_NONE = 0; public static final int DATA_CONNECTION_GPRS = 1; public static final int DATA_CONNECTION_EDGE = 2; public static final int DATA_CONNECTION_UMTS = 3; public static final int DATA_CONNECTION_OTHER = 4; /** * Returns the time in microseconds that wifi has been on while the device was * running on battery. * * {@hide} */

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	public abstract long getWifiOnTime(long batteryRealtime, int which); /** * Returns the time in microseconds that bluetooth has been on while the device was * running on battery. * * {@hide} */ public abstract long getBluetoothOnTime(long batteryRealtime, int which); /** * Return whether we are currently running on battery. */ public abstract boolean getIsOnBattery(); /** * Returns the time that the radio was on for data transfers. * @return the uptime in microseconds while unplugged */ public abstract long getRadioDataUptime(); /** * Returns the current battery realtime in microseconds. * * @param curTime the amount of elapsed realtime in microseconds. */ public abstract long getBatteryRealtime(long curTime); /**

'184 Claims	Android Device with One or More Apps
	* Returns the battery percentage level at the last time the device was unplugged from power, or * the last time it booted on battery power. */ public abstract int getDischargeStartLevel(); **Android 2.2** **SAMSUNG_PRIORART0005353, Power** /** * Brightness value to use when battery is low */ public static final int BRIGHTNESS_LOW_BATTERY = 10; /** * Threshold for BRIGHTNESS_LOW_BATTERY (percentage) * Screen will stay dim if battery level is <= LOW_BATTERY_THRESHOLD */ public static final int LOW_BATTERY_THRESHOLD = 10; /** * Turn the screen on or off * * @param on Whether you want the screen on or off */ public static native int setScreenState(boolean on); **SAMSUNG_PRIORART0005353, BatteryManager** /**

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* Extra for {@link android.content.Intent#ACTION_BATTERY_CHANGED}: * integer indicating whether the device is plugged in to a power * source; 0 means it is on battery, other constants are different * types of power sources. */ public static final String EXTRA_PLUGGED = "plugged"; /** * Extra for {@link android.content.Intent#ACTION_BATTERY_CHANGED}: * integer containing the current battery voltage level. */ public static final String EXTRA_VOLTAGE = "voltage"; /** * Extra for {@link android.content.Intent#ACTION_BATTERY_CHANGED}: * integer containing the current battery temperature. */ public static final String EXTRA_TEMPERATURE = "temperature"; /** * Extra for {@link android.content.Intent#ACTION_BATTERY_CHANGED}: * String describing the technology of the current battery. */ public static final String EXTRA_TECHNOLOGY = "technology"; // values for "status" field in the ACTION_BATTERY_CHANGED Intent public static final int BATTERY_STATUS_UNKNOWN = 1; public static final int BATTERY_STATUS_CHARGING = 2; public static final int BATTERY_STATUS_DISCHARGING = 3; public static final int BATTERY_STATUS_NOT_CHARGING = 4;

'184 Claims	Android Device with One or More Apps

	public static final int BATTERY_STATUS_FULL = 5;
	
	// values of the "plugged" field in the ACTION_BATTERY_CHANGED intent.
	// These must be powers of 2.
	/** Power source is an AC charger. */
	public static final int BATTERY_PLUGGED_AC = 1;
	/** Power source is a USB port. */
	public static final int BATTERY_PLUGGED_USB = 2;
	
	SAMSUNG_PRIORART0005353, BatteryStats
	
	/**
	* A class providing access to battery usage statistics, including information on
	* wakelocks, processes, packages, and services. All times are represented in microseconds
	* except where indicated otherwise.
	* @hide
	*/
	
	/**
	* A constant indicating a a wifi turn on timer
	*
	* {@hide}
	*/
	public static final int WIFI_TURNED_ON = 4;
	/**
	* A constant indicating an audio turn on timer
	*
	* {@hide}
	*/

'184 Claims	Android Device with One or More Apps

	public static final int AUDIO_TURNED_ON = 7;
	
	/**
	* A constant indicating a video turn on timer
	*
	* {@hide}
	*/
	public static final int VIDEO_TURNED_ON = 8;
	
	public static final int SIGNAL_STRENGTH_NONE_OR_UNKNOWN = 0;
	public static final int SIGNAL_STRENGTH_POOR = 1;
	public static final int SIGNAL_STRENGTH_MODERATE = 2;
	public static final int SIGNAL_STRENGTH_GOOD = 3;
	public static final int SIGNAL_STRENGTH_GREAT = 4;
	
	static final String[] SIGNAL_STRENGTH_NAMES = {
	"none", "poor", "moderate", "good", "great"
	};
	
	public static final int DATA_CONNECTION_NONE = 0;
	public static final int DATA_CONNECTION_GPRS = 1;
	public static final int DATA_CONNECTION_EDGE = 2;
	public static final int DATA_CONNECTION_UMTS = 3;
	public static final int DATA_CONNECTION_OTHER = 4;
	
	GOOG-HEADWATER-00000092, Google I/O 2009 - Coding for Life -- Battery Life, That Is (June 2, 2009)

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

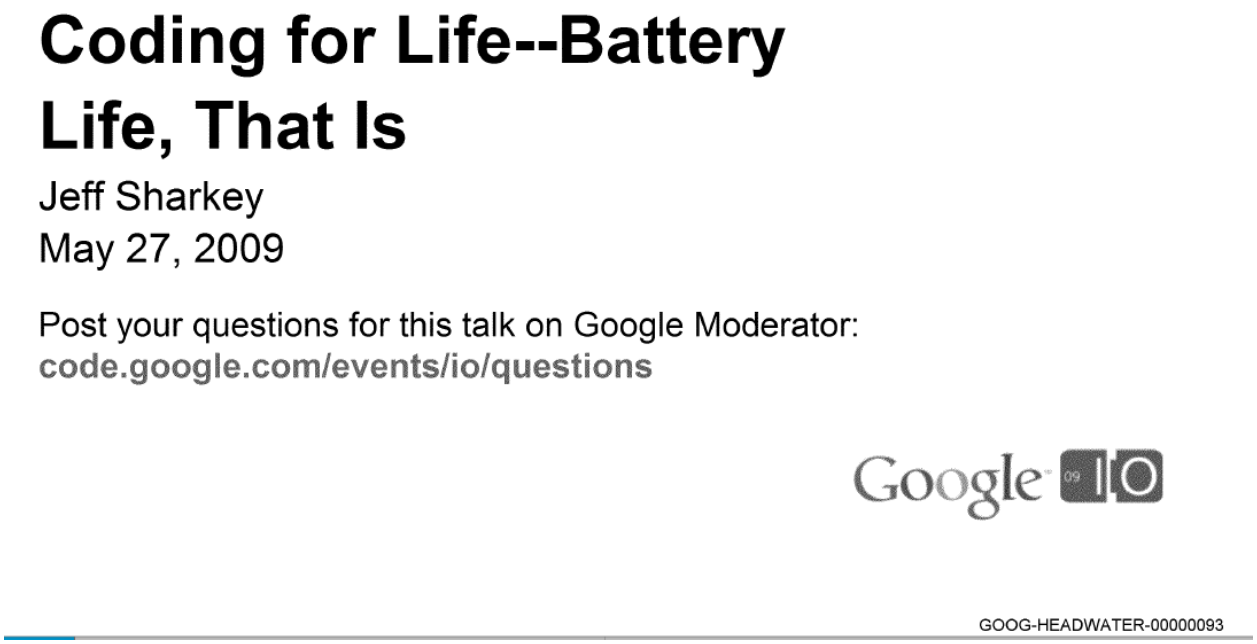
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 2: 

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 3: Why does this matter? ● Phones primarily run on battery power, and each device has a "**battery budget**" ○ When it's gone, it's gone ○ Apps need to work together to be good citizens of that shared resource ○ Current measured in mA, battery capacity in mAh ● HTC Dream: **1150mAh** ● HTC Magic: **1350mAh** ● Samsung I7500: **1500mAh** ● Asus Eee PC: **5800mAh** Google 09 I/O GOOG-HEADWATER-00000094

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

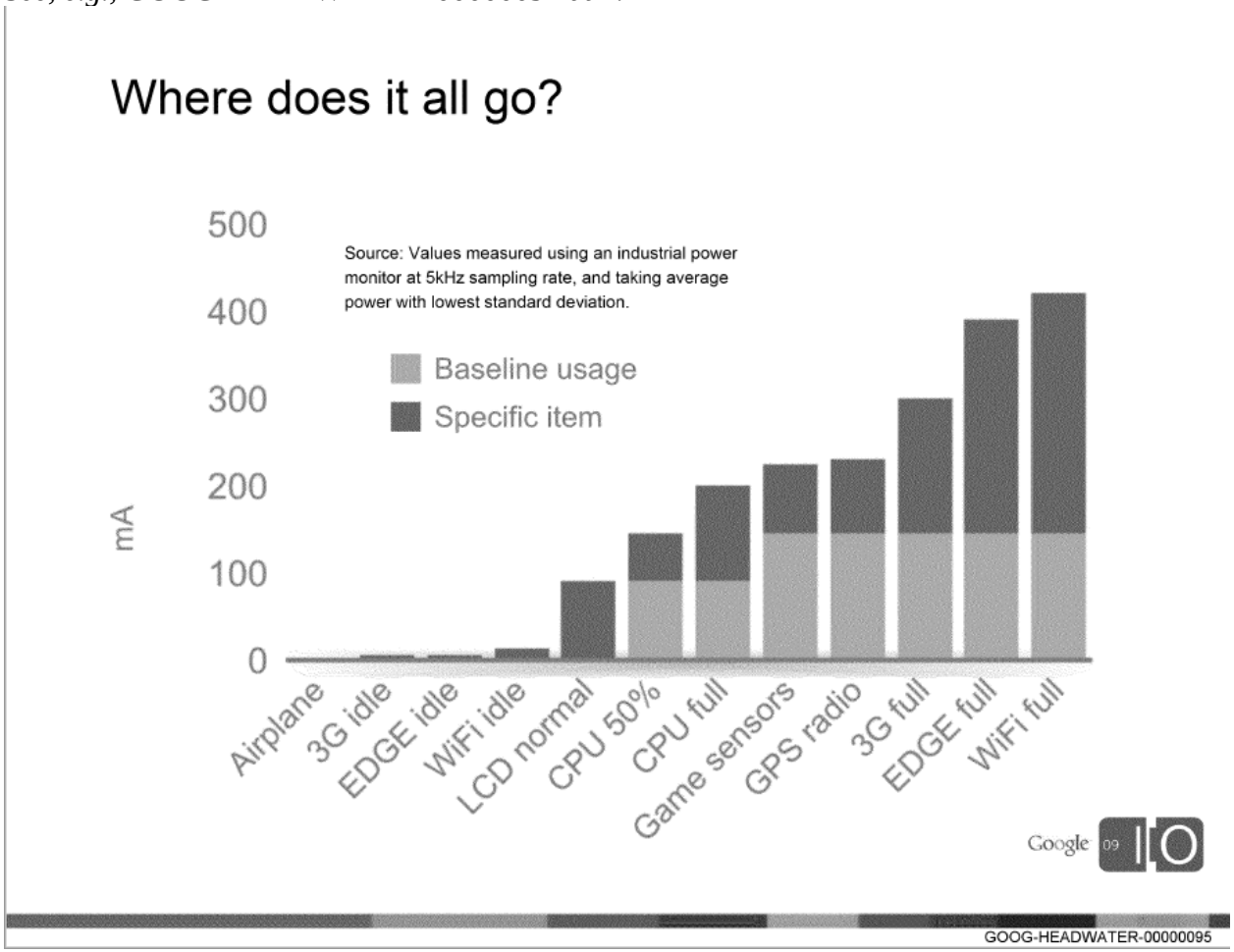
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 4: 

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

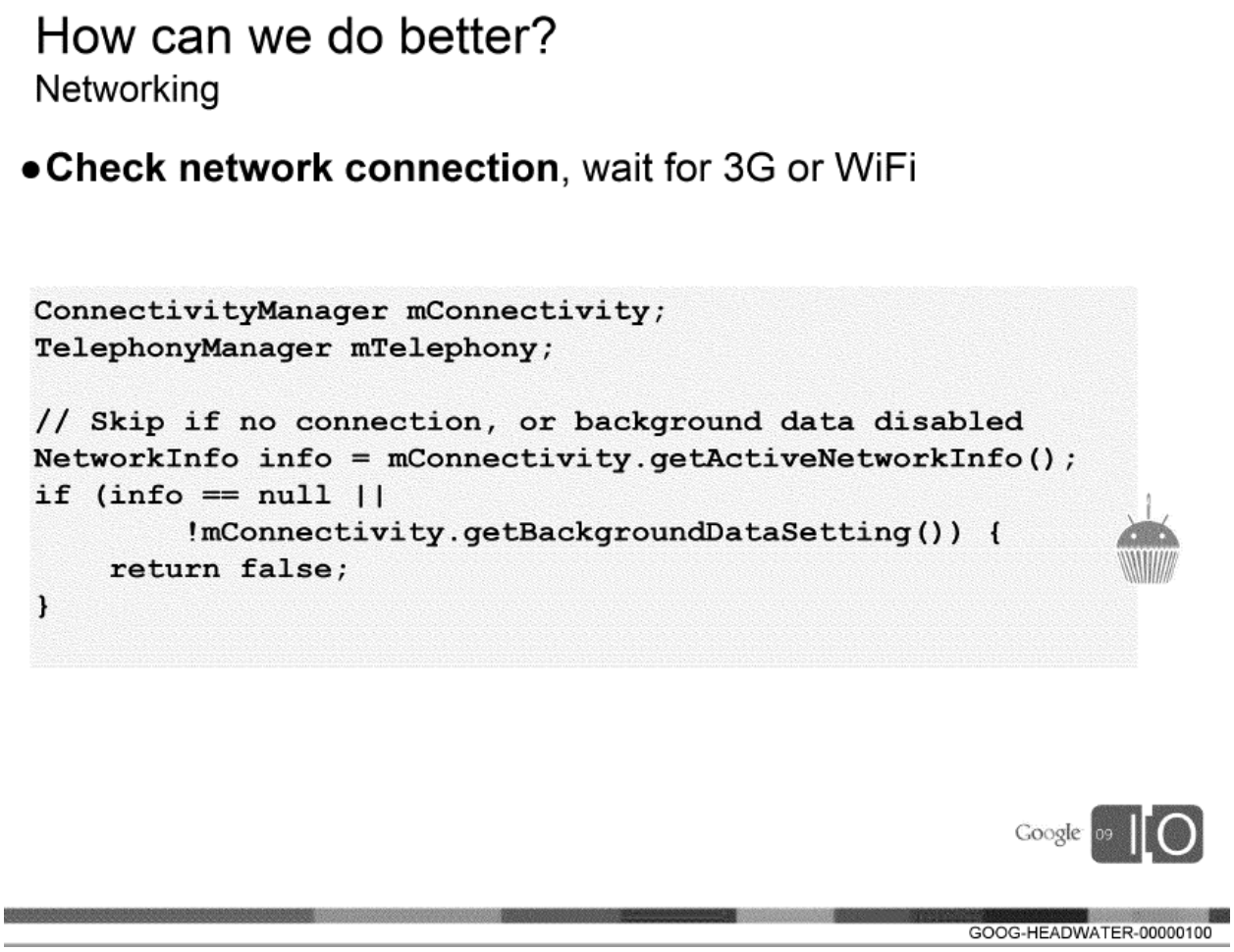
'184 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 9:  *See, e.g.*, GOOG-HEADWATER-00000092 at 11:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	How can we do better?
	Networking
	● **Check network connection**, wait for 3G or WiFi
	`// Only update if WiFi or 3G is connected and not roaming`
	`int netType = info.getType();`
	`int netSubtype = info.getSubtype();`
	`if (netType == ConnectivityManager.TYPE_WIFI) {`
	`  return info.isConnected();`
	`} else if (netType == ConnectivityManager.TYPE_MOBILE`
	`  && netSubtype == TelephonyManager.NETWORK_TYPE_UMTS`
	`  && !mTelephony.isNetworkRoaming()) {`
	`  return info.isConnected();`
	`} else {`
	`  return false;`
	`}`
	Google 09 I/O
	GOOG-HEADWATER-00000101
	See, e.g., GOOG-HEADWATER-00000092 at 16:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

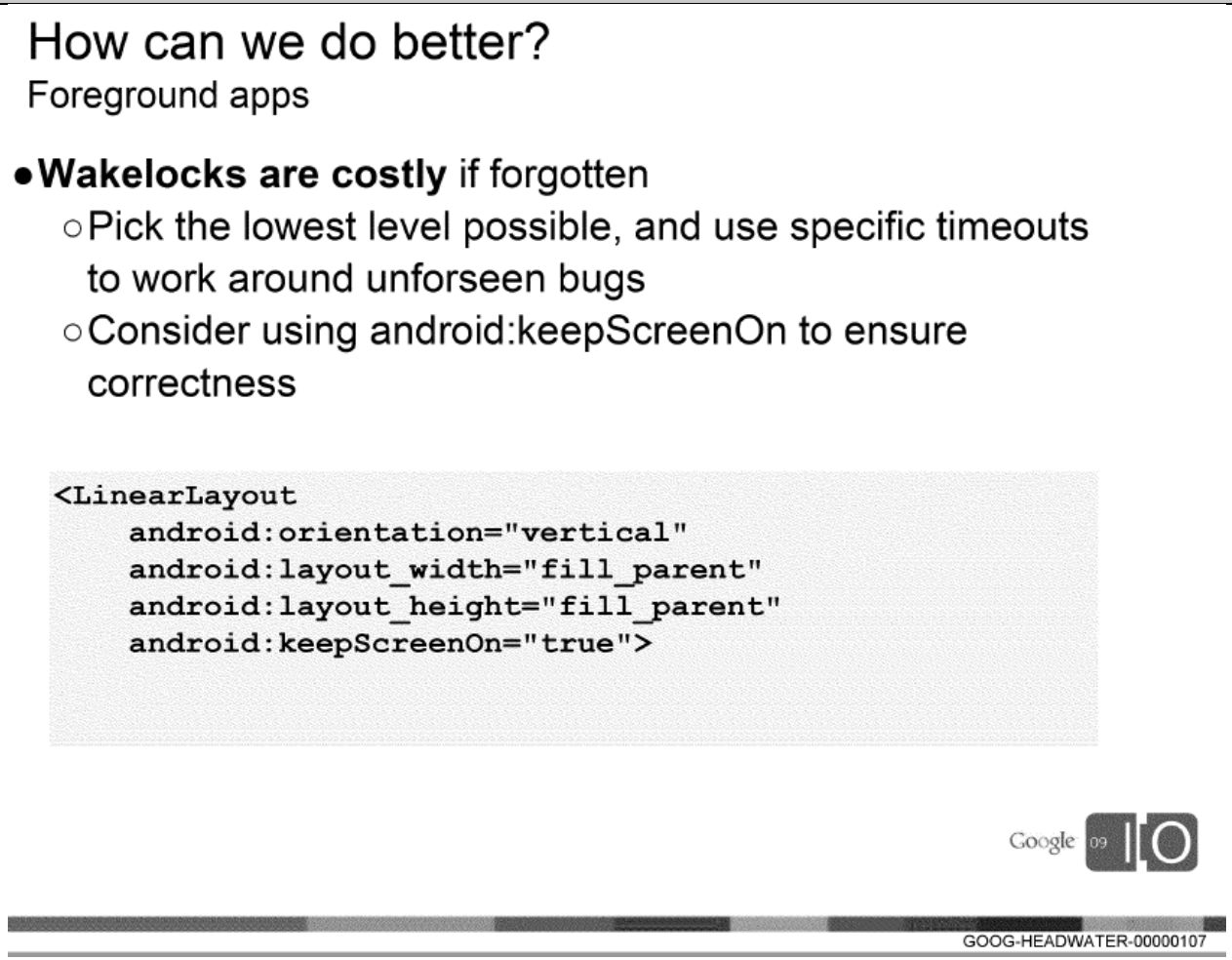
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 18:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

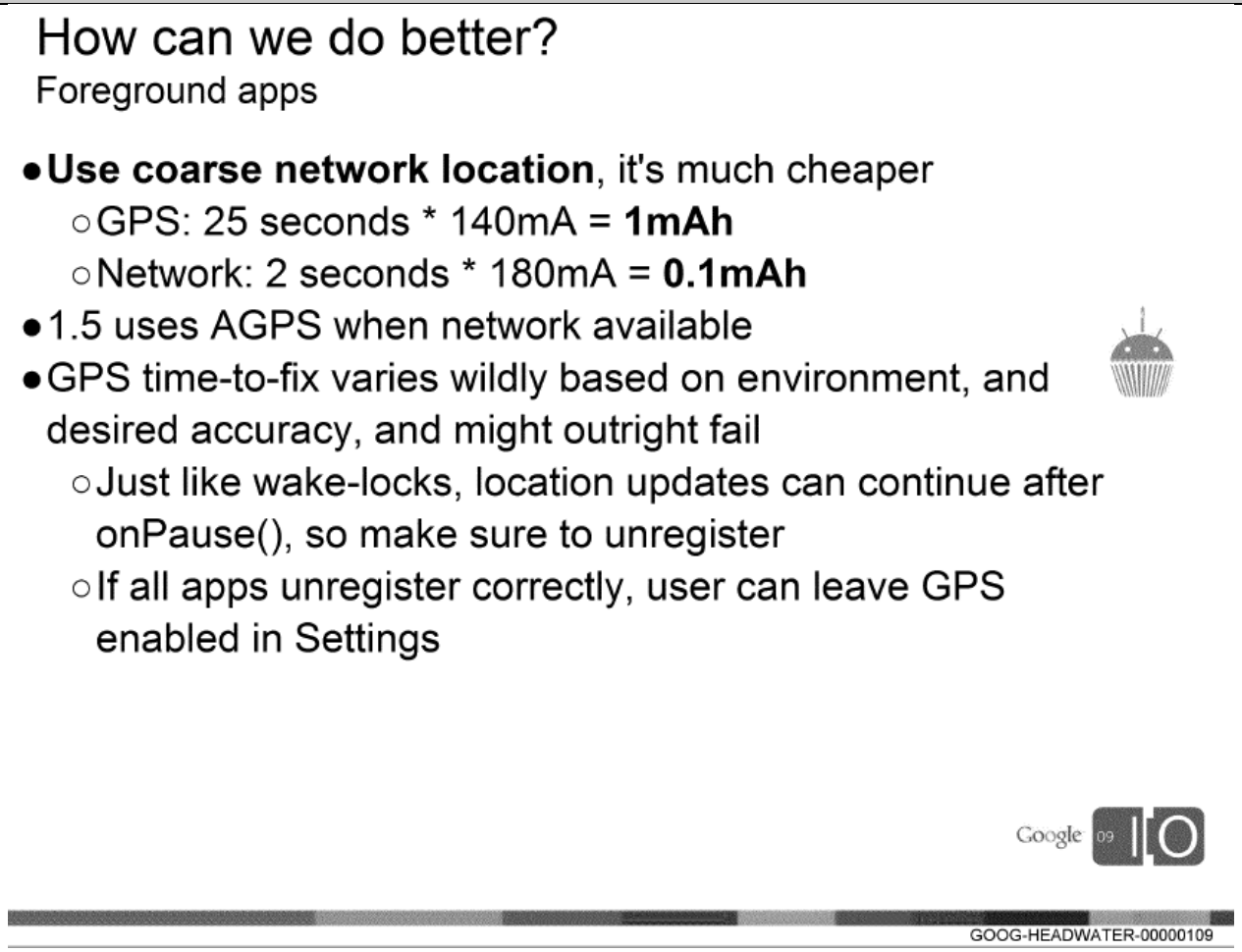
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 20:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

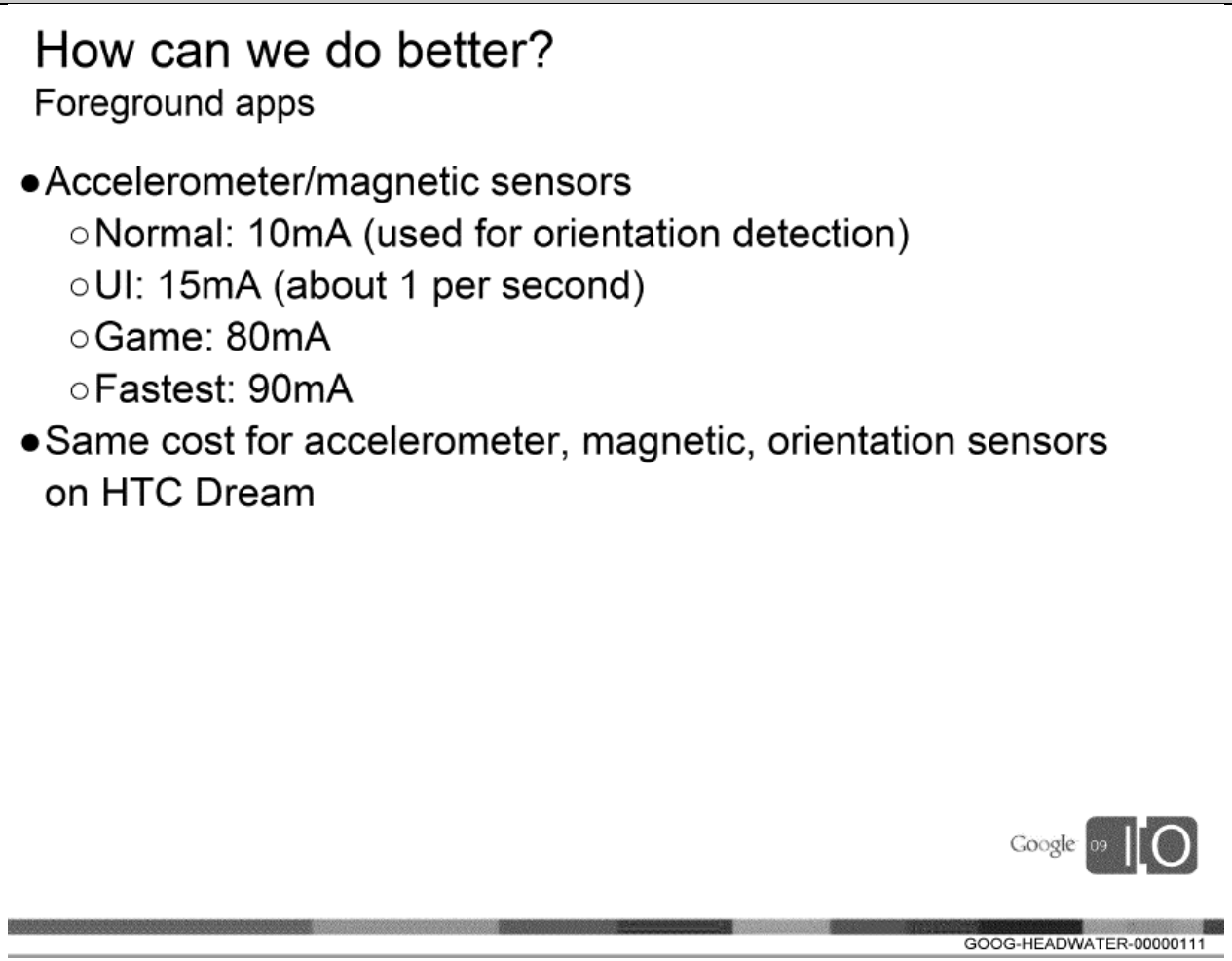
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 22:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

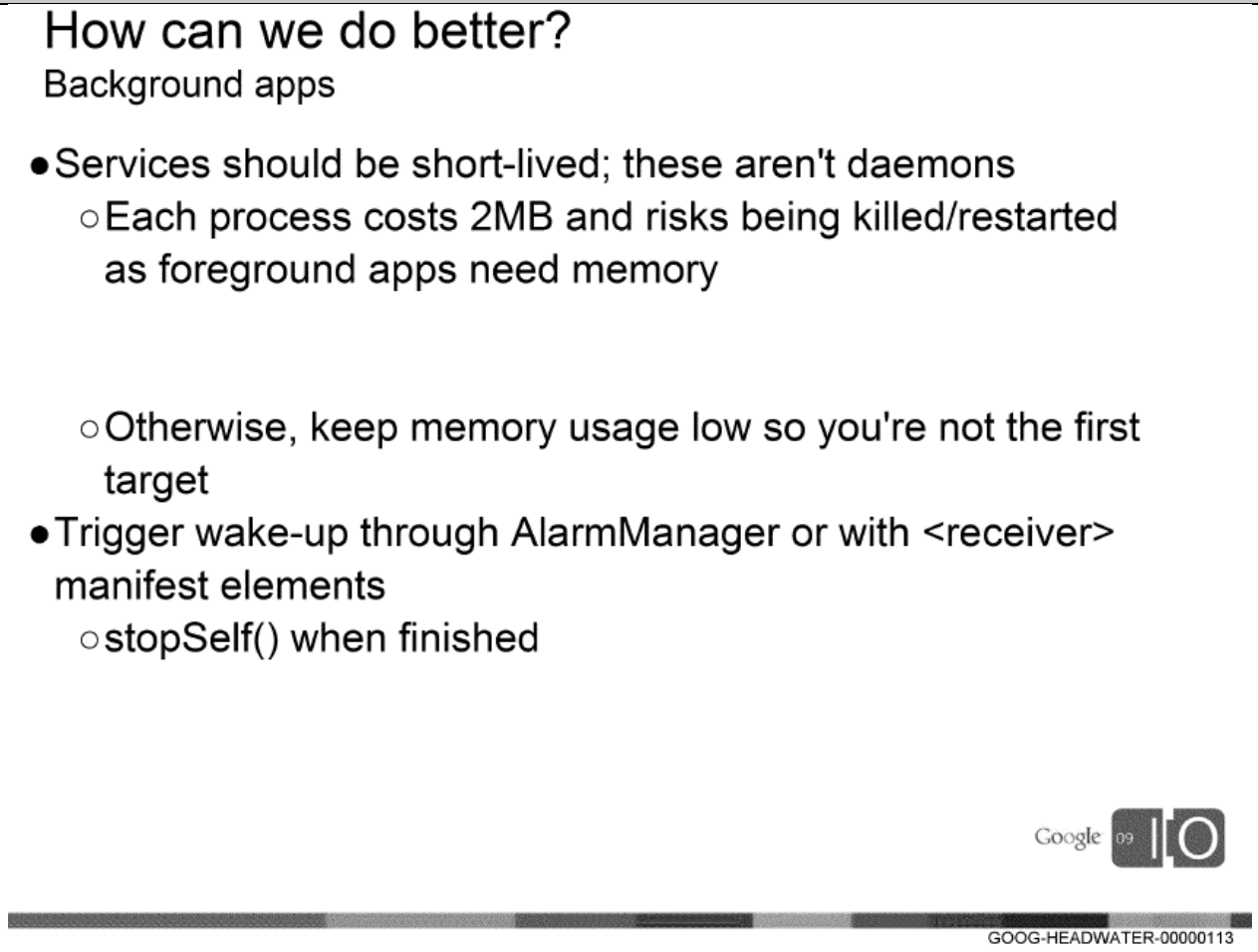
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 26:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

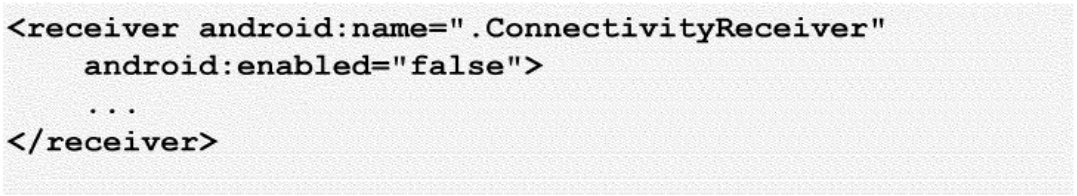
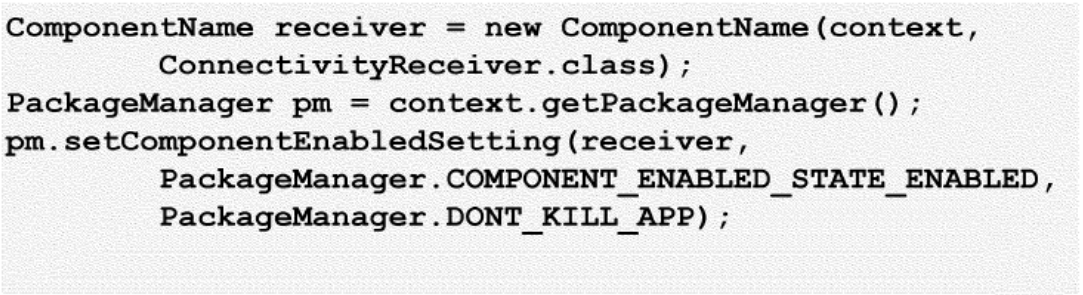
'184 Claims	Android Device with One or More Apps
	How can we do better? Background apps ● Dynamically enabling/disabling <receiver> components in manifest, especially when no-ops   Google 09 I/O GOOG-HEADWATER-00000117

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	See, e.g., GOOG-HEADWATER-00000092 at 27:
	How can we do better?
	Background apps
	● Checking current battery and network state before running a full update
	`public void onCreate() {`
	`  // Register for sticky broadcast and send default`
	`  registerReceiver(mReceiver, mFilter);`
	`  mHandler.sendEmptyMessageDelayed(MSG_BATT, 1000);`
	`}`
	`IntentFilter mFilter =`
	`  new IntentFilter(Intent.ACTION_BATTERY_CHANGED);`
	`BroadcastReceiver mReceiver = new BroadcastReceiver() {`
	`  public void onReceive(Context context, Intent intent) {`
	`  // Found sticky broadcast, so trigger update`
	`  unregisterReceiver(mReceiver);`
	`  mHandler.removeMessages(MSG_BATT);`
	`  mHandler.obtainMessage(MSG_BATT, intent).sendToTarget();`
	`  }`
	`};`
	Google 09 I/O
	GOOG-HEADWATER-00000118
	See, e.g., GOOG-HEADWATER-00000092 at 29:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

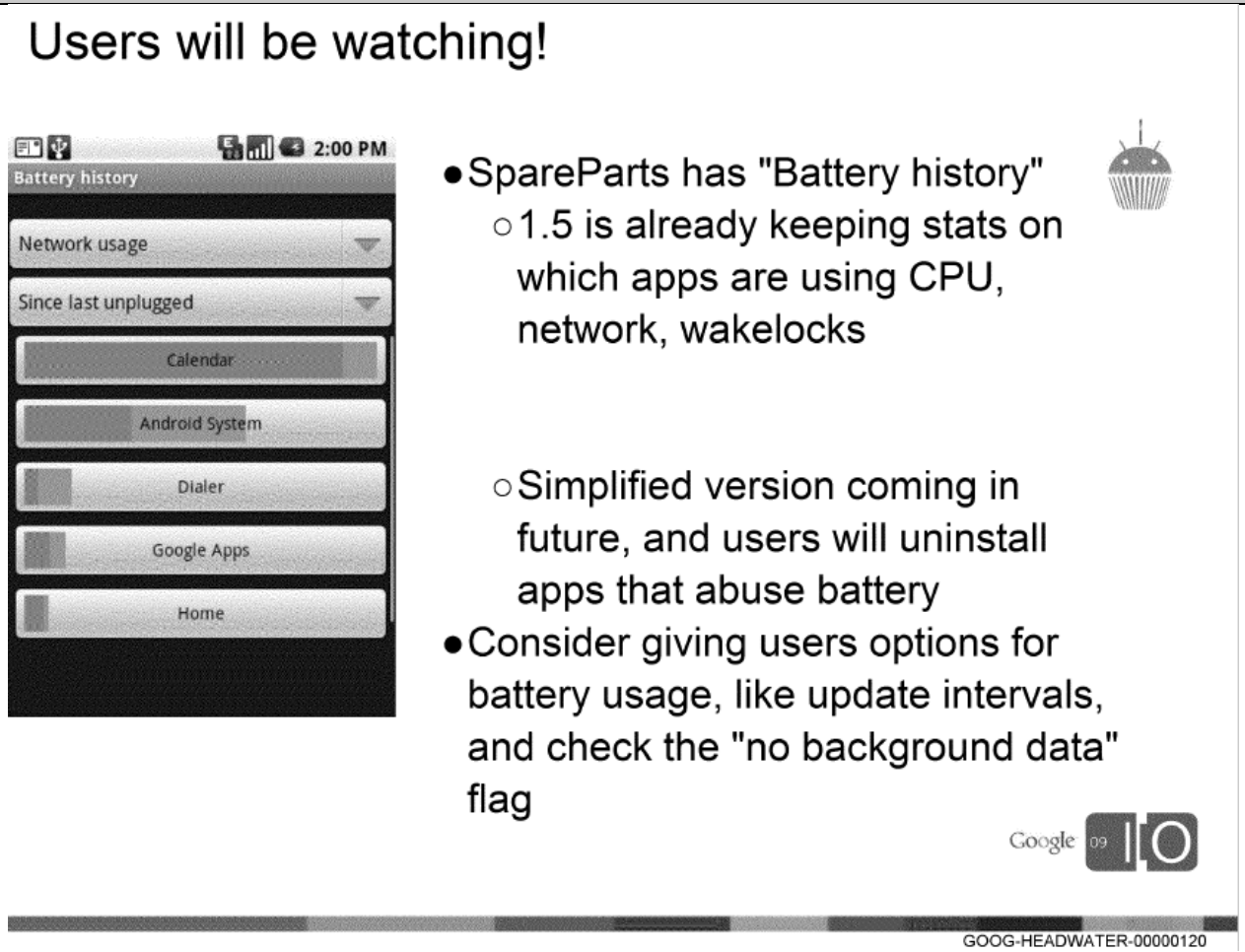
'184 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 30:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	Takeaways ● Use an efficient parser and GZIP to make best use of network and CPU resources ● Services that sleep or poll are bad, use <receiver> and AlarmManager instead ○ Disable manifest elements when no-op ○ Wake up along with everyone else (inexact alarms) ● Wait for better network/battery for bulk transfers ● Give users choices about background behavior Google I/O 09 GOOG-HEADWATER-00000121

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
[11] The wireless end-user device of claim 1, wherein the one or more processors are further configured to dynamically change the determination of whether to apply the differential traffic control policy based on a device usage state.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d], [10].
[12] The wireless end-user device of claim 1, wherein the one or more processors are further configured to dynamically change the determination of whether to apply the differential traffic control policy based on power control state changes for the WWAN modem.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d], [10].
[13] The wireless end-user device of claim 1, wherein the one or more processors are further	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures:

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
configured to update at least a portion of the differential traffic control policy and/or the differential traffic control policy list based on information received from a network element.	*See, e.g.*, the disclosures identified for claims [1 pre] – [1d].
[14] The wireless end-user device of claim 1, wherein the one or more processors are further configured to, when the Internet service access request is blocked, prevent the first Internet access request from causing a change to a power state of the modem.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d], [10].
[15] The wireless end-user device of claim 14, wherein the one or more processors are further configured to, when the power state of the modem is idle or sleeping, prevent power	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d], [10].

'184 Claims	**Android Device with One or More Apps**
state changes to the modem until an Internet service access request is allowed.	
[16]. The wireless end-user device of claim 1, wherein the one or more processors are further configured to, when the Internet service access request is blocked, instruct the first end-user application to transition to a different state.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d], [10].
[17] The wireless end-user device of claim 1, further comprising an agent to block, modify, remove, or replace user interface messages generated by the particular application based on the block of the Internet service access request.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d]. In addition, *see, e.g.*: **Android 1.0** **SAMSUNG_PRIORART0005487, Socket.cpp** #include <utils/Socket.h> #include <utils/inet_address.h> #include <utils/Log.h>

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	#include <utils/Timers.h> #ifndef HAVE_WINSOCK # include <sys/types.h> # include <sys/socket.h> # include <netinet/in.h> # include <arpa/inet.h> #endif #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <string.h> #include <errno.h> #include <assert.h> using namespace android; /* * ========== * Socket * ========== */ #ifndef INVALID_SOCKET # define INVALID_SOCKET (-1) #endif #define UNDEF_SOCKET ((unsigned long) INVALID_SOCKET) /*static*/ bool Socket::mBootInitialized = false; /* * Extract system-dependent error code. */ static inline int getSocketError(void) {

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	return WSAGetLastError();
	#else
	return errno;
	#endif
	}
	/*
	* One-time initialization for socket code.
	*/
	/*static*/ bool Socket::bootInit(void)
	{
	#ifdef HAVE_WINSOCK
	WSADATA wsaData;
	int err;
	err = WSAStartup(MAKEWORD(2, 0), &wsaData);
	if (err != 0) {
	LOG(LOG_ERROR, "socket", "Unable to start WinSock\n");
	return false;
	}
	LOG(LOG_INFO, "socket", "Using WinSock v%d.%d\n",
	LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
	#endif
	mBootInitialized = true;
	return true;
	}
	/*
	* One-time shutdown for socket code.
	*/
	/*static*/ void Socket::finalShutdown(void)
	{

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	WSACleanup();
	#endif
	mBootInitialized = false;
	}
	/*
	* Simple constructor. Allow the application to create us and then make
	* bind/connect calls.
	*/
	Socket::Socket(void)
	: mSock(UNDEF_SOCKET)
	{
	if (!mBootInitialized)
	LOG(LOG_WARN, "socket", "WARNING: sockets not initialized\n");
	}
	/*
	* Destructor. Closes the socket and resets our storage.
	*/
	Socket::~Socket(void)
	{
	close();
	}
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const char* host, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	InetSocketAddress sockAddr;
	if (!sockAddr.create(host, port))
	return -1;
	//return doConnect(sockAddr);
	int foo;
	foo = doConnect(sockAddr);
	return foo;
	}
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const InetAddress* addr, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	InetSocketAddress sockAddr;
	if (!sockAddr.create(addr, port))
	return -1;
	return doConnect(sockAddr);
	}
	/*
	* Finish creating a socket by connecting to the remote host.
	*
	* Returns 0 on success.
	*/
	int Socket::doConnect(const InetSocketAddress& sockAddr)
	{

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	SOCKET sock;
	#else
	int sock;
	#endif
	const InetAddress* addr = sockAddr.getAddress();
	int port = sockAddr.getPort();
	struct sockaddr_in inaddr;
	DurationTimer connectTimer;
	assert(sizeof(struct sockaddr_in) == addr->getAddressLength());
	memcpy(&inaddr, addr->getAddress(), addr->getAddressLength());
	inaddr.sin_port = htons(port);
	//fprintf(stderr, "--- connecting to %s:%d\n",
	// sockAddr.getHostName(), port);
	sock = ::socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (sock == INVALID_SOCKET) {
	int err = getSocketError();
	LOG(LOG_ERROR, "socket", "Unable to create socket (err=%d)\n", err);
	return (err != 0) ? err : -1;
	}
	connectTimer.start();
	if (::connect(sock, (struct sockaddr*) &inaddr, sizeof(inaddr)) != 0) {
	int err = getSocketError();
	LOG(LOG_WARN, "socket", "Connect to %s:%d failed: %d\n",
	sockAddr.getHostName(), port, err);
	return (err != 0) ? err : -1;
	}
	connectTimer.stop();
	if ((long) connectTimer.durationUsecs() > 100000) {
	LOG(LOG_INFO, "socket",

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	"Connect to %s:%d took %.3fs\n", sockAddr.getHostName(),
	port, ((long) connectTimer.durationUsecs()) / 1000000.0);
	}
	mSock = (unsigned long) sock;
	LOG(LOG_VERBOSE, "socket",
	"--- connected to %s:%d\n", sockAddr.getHostName(), port);
	return 0;
	}
	/*
	* Close the socket if it needs closing.
	*/
	bool Socket::close(void)
	{
	if (mSock != UNDEF_SOCKET) {
	//fprintf(stderr, "--- closing socket %lu\n", mSock);
	#ifdef HAVE_WINSOCK
	if (::closesocket((SOCKET) mSock) != 0)
	return false;
	#else
	if (::close((int) mSock) != 0)
	return false;
	#endif
	}
	mSock = UNDEF_SOCKET;
	return true;
	}
	/*
	* Read data from socket.
	*
	* Standard semantics: read up to "len" bytes into "buf". Returns the

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	* number of bytes read, or less than zero on error.
	*/
	int Socket::read(void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: read on invalid socket\n");
	return -500;
	}
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	cc = recv(sock, (char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}
	return cc;
	}
	/*
	* Write data to a socket.
	*
	* Standard semantics: write up to "len" bytes into "buf". Returns the
	* number of bytes written, or less than zero on error.
	*/
	int Socket::write(const void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {

'184 Claims	Android Device with One or More Apps

	LOG(LOG_ERROR, "socket", "ERROR: write on invalid socket\n");
	return -500;
	}
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	cc = send(sock, (const char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}
	return cc;
	
	Android 1.6
	
	SAMSUNG_PRIORART0005350, Socket.cpp
	
	//
	// Internet address class.
	//
	
	#ifdef HAVE_WINSOCK
	// This needs to come first, or Cygwin gets concerned about a potential
	// clash between WinSock and <sys/types.h>.
	# include <winsock2.h>
	#endif

'184 Claims	Android Device with One or More Apps
	#include <utils/Socket.h> #include <utils/inet_address.h> #include <utils/Log.h> #include <utils/Timers.h> #ifndef HAVE_WINSOCK # include <sys/types.h> # include <sys/socket.h> # include <netinet/in.h> # include <arpa/inet.h> #endif #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <string.h> #include <errno.h> #include <assert.h> using namespace android; /* * ========== * Socket * ========== */

'184 Claims	Android Device with One or More Apps

	#ifndef INVALID_SOCKET
	# define INVALID_SOCKET (-1)
	#endif
	#define UNDEF_SOCKET ((unsigned long) INVALID_SOCKET)
	
	/*static*/ bool Socket::mBootInitialized = false;
	
	/*
	* Extract system-dependent error code.
	*/
	static inline int getSocketError(void) {
	#ifdef HAVE_WINSOCK
	return WSAGetLastError();
	#else
	return errno;
	#endif
	}
	
	/*
	* One-time initialization for socket code.
	*/
	/*static*/ bool Socket::bootInit(void)
	{
	#ifdef HAVE_WINSOCK
	WSADATA wsaData;
	int err;
	
	err = WSAStartup(MAKEWORD(2, 0), &wsaData);
	if (err != 0) {

'184 Claims	Android Device with One or More Apps

	LOG(LOG_ERROR, "socket", "Unable to start WinSock\n");
	return false;
	}
	
	LOG(LOG_INFO, "socket", "Using WinSock v%d.%d\n",
	LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
	#endif
	
	mBootInitialized = true;
	return true;
	}
	
	/*
	* One-time shutdown for socket code.
	*/
	/*static*/ void Socket::finalShutdown(void)
	{
	#ifdef HAVE_WINSOCK
	WSACleanup();
	#endif
	mBootInitialized = false;
	}
	
	/*
	* Simple constructor. Allow the application to create us and then make
	* bind/connect calls.
	*/
	Socket::Socket(void)
	: mSock(UNDEF_SOCKET)

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	{
	if (!mBootInitialized)
	LOG(LOG_WARN, "socket", "WARNING: sockets not initialized\n");
	}
	
	/*
	* Destructor. Closes the socket and resets our storage.
	*/
	Socket::~Socket(void)
	{
	close();
	}
	
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const char* host, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	
	InetSocketAddress sockAddr;
	if (!sockAddr.create(host, port))
	return -1;
	
	//return doConnect(sockAddr);
	int foo;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	foo = doConnect(sockAddr);
	return foo;
	}
	
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const InetAddress* addr, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	
	InetSocketAddress sockAddr;
	if (!sockAddr.create(addr, port))
	return -1;
	
	return doConnect(sockAddr);
	}
	
	/*
	* Finish creating a socket by connecting to the remote host.
	*
	* Returns 0 on success.
	*/
	int Socket::doConnect(const InetSocketAddress& sockAddr)
	{
	#ifdef HAVE_WINSOCK
	SOCKET sock;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	#else
	int sock;
	#endif
	const InetAddress* addr = sockAddr.getAddress();
	int port = sockAddr.getPort();
	struct sockaddr_in inaddr;
	DurationTimer connectTimer;
	
	assert(sizeof(struct sockaddr_in) == addr->getAddressLength());
	memcpy(&inaddr, addr->getAddress(), addr->getAddressLength());
	inaddr.sin_port = htons(port);
	
	//fprintf(stderr, "--- connecting to %s:%d\n",
	// sockAddr.getHostName(), port);
	
	sock = ::socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (sock == INVALID_SOCKET) {
	int err = getSocketError();
	LOG(LOG_ERROR, "socket", "Unable to create socket (err=%d)\n", err);
	return (err != 0) ? err : -1;
	}
	
	connectTimer.start();
	
	if (::connect(sock, (struct sockaddr*) &inaddr, sizeof(inaddr)) != 0) {
	int err = getSocketError();
	LOG(LOG_WARN, "socket", "Connect to %s:%d failed: %d\n",
	sockAddr.getHostName(), port, err);
	return (err != 0) ? err : -1;
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	connectTimer.stop();
	if ((long) connectTimer.durationUsecs() > 100000) {
	LOG(LOG_INFO, "socket",
	"Connect to %s:%d took %.3fs\n", sockAddr.getHostName(),
	port, ((long) connectTimer.durationUsecs()) / 1000000.0);
	}
	
	mSock = (unsigned long) sock;
	LOG(LOG_VERBOSE, "socket",
	"--- connected to %s:%d\n", sockAddr.getHostName(), port);
	return 0;
	}
	
	/*
	* Close the socket if it needs closing.
	*/
	bool Socket::close(void)
	{
	if (mSock != UNDEF_SOCKET) {
	//fprintf(stderr, "--- closing socket %lu\n", mSock);
	#ifdef HAVE_WINSOCK
	if (::closesocket((SOCKET) mSock) != 0)
	return false;
	#else
	if (::close((int) mSock) != 0)
	return false;
	#endif
	}

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	mSock = UNDEF_SOCKET;
	
	return true;
	}
	
	/*
	* Read data from socket.
	*
	* Standard semantics: read up to "len" bytes into "buf". Returns the
	* number of bytes read, or less than zero on error.
	*/
	int Socket::read(void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: read on invalid socket\n");
	return -500;
	}
	
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	
	cc = recv(sock, (char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps

	}
	
	return cc;
	}
	
	/*
	* Write data to a socket.
	*
	* Standard semantics: write up to "len" bytes into "buf". Returns the
	* number of bytes written, or less than zero on error.
	*/
	int Socket::write(const void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: write on invalid socket\n");
	return -500;
	}
	
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	
	cc = send(sock, (const char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}

'184 Claims	Android Device with One or More Apps
	return cc; } **Android 2.2** **SAMSUNG_PRIORART0005060, Socket** /** * Creates a new unconnected socket. When a SocketImplFactory is defined it * creates the internal socket implementation, otherwise the default socket * implementation will be used for this socket. * * @see SocketImplFactory * @see SocketImpl */ /** * Tries to connect a socket to all IP addresses of the given hostname. * * @param dstName * the target host name or IP address to connect to. * @param dstPort * the port on the target host to connect to. * @param localAddress * the address on the local host to bind to. * @param localPort * the port on the local host to bind to. * @param streaming

'184 Claims	Android Device with One or More Apps
	* if {@code true} a streaming socket is returned, a datagram * socket otherwise. * @throws UnknownHostException * if the host name could not be resolved into an IP address. * @throws IOException * if an error occurs while creating the socket. * @throws SecurityException * if a security manager exists and it denies the permission to * connect to the given address and port. */ /** * Creates a new streaming socket connected to the target host specified by * the parameters {@code dstName} and {@code dstPort}. The socket is bound * to any available port on the local host. * <p>Implementation note: this implementation tries each * IP address for the given hostname until it either connects successfully * or it exhausts the set. It will try both IPv4 and IPv6 addresses in the * order specified by the system property {@code "java.net.preferIPv6Addresses"}. * * @param dstName * the target host name or IP address to connect to. * @param dstPort * the port on the target host to connect to. * @throws UnknownHostException * if the host name could not be resolved into an IP address. * @throws IOException * if an error occurs while creating the socket. * @throws SecurityException * if a security manager exists and it denies the permission to * connect to the given address and port.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	*/ /** * Checks whether the connection destination satisfies the security policy * and the validity of the port range. * * @param destAddr * the destination host address. * @param dstPort * the port on the destination host. */ /** * Closes the socket. It is not possible to reconnect or rebind to this * socket thereafter which means a new socket instance has to be created. * * @throws IOException * if an error occurs while closing the socket. */ /** * Gets the IP address of the target host this socket is connected to. * * @return the IP address of the connected target host or {@code null} if * this socket is not yet connected. */ /** * Gets the local IP address this socket is bound to. * * @return the local IP address of this socket or {@code InetAddress.ANY} if

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* the socket is unbound. */ /** * Gets the local port this socket is bound to. * * @return the local port of this socket or {@code -1} if the socket is * unbound. */ /** * Gets the port number of the target host this socket is connected to. * * @return the port number of the connected target host or {@code 0} if this * socket is not yet connected. */ /** * Gets the local address and port of this socket as a SocketAddress or * {@code null} if the socket is unbound. This is useful on multihomed * hosts. * * @return the bound local socket address and port. */ /** * Binds this socket to the given local host address and port specified by * the SocketAddress {@code localAddr}. If {@code localAddr} is set to * {@code null}, this socket will be bound to an available local address on * any free port.

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	Android Device with One or More Apps
	* * @param localAddr * the specific address and port on the local machine to bind to. * @throws IllegalArgumentException * if the given SocketAddress is invalid or not supported. * @throws IOException * if the socket is already bound or an error occurs while * binding. */ /** * Connects this socket to the given remote host address and port specified * by the SocketAddress {@code remoteAddr}. * * @param remoteAddr * the address and port of the remote host to connect to. * @throws IllegalArgumentException * if the given SocketAddress is invalid or not supported. * @throws IOException * if the socket is already connected or an error occurs while * connecting. */
[18] The wireless end-user device of claim 1, wherein the one or more processors operable to block the Internet service access request	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] – [1d], [17].

'184 Claims	**Android Device with One or More Apps**
are configured to selectively block the Internet service access request by intercepting open, connect, and/or write requests by the particular application to a network stack.	
[19] The wireless end-user device of claim 18, wherein the one or more processors are configured to respond to an intercepted request by the particular application by emulating network messaging.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d], [17].
[20]. The wireless end-user device of claim 19, wherein emulating network messaging comprises responding to a network request from the particular application by blocking the request from passing to a network stack and	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] – [1d], [17].

Exhibit B-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,271,184

'184 Claims	**Android Device with One or More Apps**
returning to the particular application a message indicating the network request was not successful.	